数据结构习题解答与考试指导

梁作娟　胡 伟　唐瑞春　编著

清 华 大 学 出 版 社
北　京

内 容 简 介

本书可作为“数据结构”课程的复习参考书，其内容、要点、题目都是根据该课程的范围和难度来组织的。

全书共分 13 章。前 12 章每章的内容编排为基本知识结构图、知识点、习题及参考答案、考研真题分析、自测题 5 个部分。最后一章精选了近几年的各大名校硕士研究生入学考试题，并附有参考答案。该书内容充实完整，层次分明，概念着眼点细微，知识点清晰，表述方式易于接受，有利于加深学生对“数据结构”课程的理解，帮助学生从广度和深度上把握知识体系，拓宽解题思路，提高解题速度。

本书针对性强，习题覆盖面广，可作为计算机及相关专业“数据结构”课程的学习指导书，也适用于报考硕士研究生的人员作为应试指导书。

图书在版编目（CIP）数据

数据结构习题解答与考试指导/梁作娟，胡伟，唐瑞春编著.
—北京：清华大学出版社，2004.11

ISBN 7-302-09820-4

Ⅰ. 数… Ⅱ. ①梁… ②胡… ③唐… Ⅲ. 数据结构—研究生—入学考试—解题
Ⅳ. TP311.12-44

中国版本图书馆 CIP 数据核字（2004）第 111839 号

出 版 者：清华大学出版社　　地　　址：北京清华大学学研大厦
　　http://www.tup.com.cn　　邮　　编：100084
社 总 机：010-62770175　　客户服务：010-62776969
组稿编辑：科海
文稿编辑：陈洁
封面设计：付剑飞
版式设计：科海
印 刷 者：北京科普瑞印刷有限责任公司
发 行 者：新华书店总店北京发行所
开　　本：787×1092　1/16　　印张：27.25　　字数：663 千字
版　　次：2004 年 11 月第 1 版　　2004 年 11 月第 1 次印刷
书　　号：ISBN 7-302-09820-4/TP・6774
印　　数：1~5000
定　　价：35.00 元

前　言

“ 数据结构 ”课程是计算机及相关专业最重要的专业基础课程之一，是进行程序设计的理论和技术基础，同时对于计算机专业其他课程的学习也是十分重要的。对于广大学生来说，“ 数据结构 ”是一门非常有用而难学的课程。本书是由长期在一线从事教学工作、有着丰富授课经验的教师编写而成。

本书的部分内容是与经典教材《数据结构（C语言版）》（清华大学出版社出版，严蔚敏、吴伟民编著）和《数据结构题集（C语言版）》（清华大学出版社出版，严蔚敏、吴伟民、米宁编著）配套使用的。

全书共分13章，分别为绪论、线性表、栈和队列、串、数组和广义表、树和二叉树、图、动态存储管理、查找、内部排序、外部排序、文件、名校试题。

前12章每章的内容编排为基本知识结构图、知识点、习题及参考答案、考研真题分析、自测题5个部分。“ 基本知识结构图 ”用框图的形式表示，可帮助学生清晰地归纳本章内容；“ 知识点 ”给出了本章应该掌握的主要知识要点；“ 习题及参考答案 ”部分给出了《数据结构题集（C语言版）》（清华大学出版社出版，严蔚敏、吴伟民、米宁编著）中的部分习题及答案，并对较难解的题目进行了分析指导，通过这些习题，可以掌握知识点中的概念，并对基础概念能有所提升；“ 考研真题分析 ”部分精心选择了近几年来的考研真题并给出分析解答；“ 自测题 ”部分安排了难度适宜的题目供学生自测，以检验自己的水平。

最后一章“ 名校试题 ”中提供了几所名牌大学2002年和2003年的硕士研究生入学考试试题，读者可以进行实战演练。

该书内容充实完整，层次分明，概念着眼点细微，知识点清晰，表述方式易于接受，有利于加深学生对“ 数据结构 ”课程的理解，帮助学生从广度和深度上把握知识体系，拓宽解题思路，提高解题速度。

由于作者水平有限，时间仓促，书中出现的一些不足和差错之处在所难免，敬请广大读者批评指正。

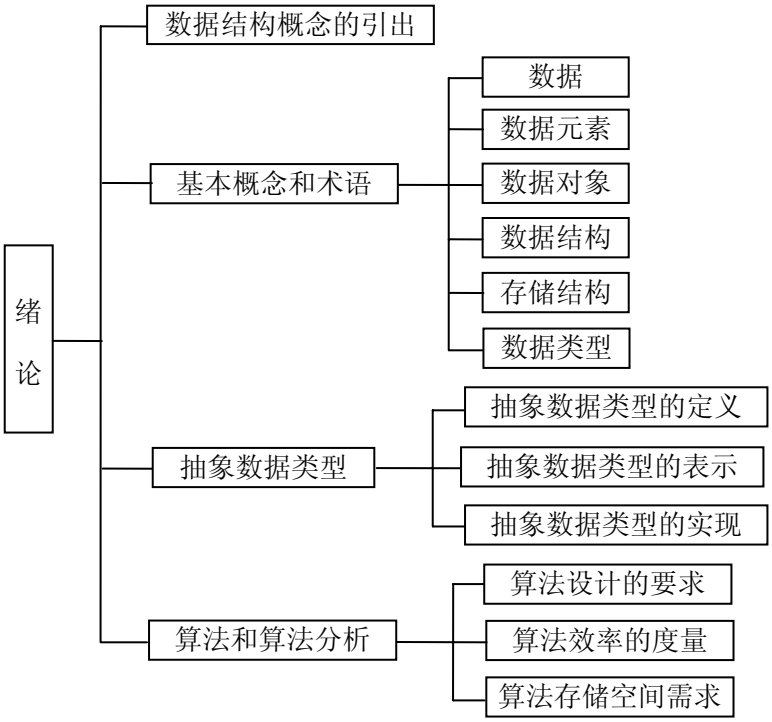
作者

2004年10月

目　　录

第1章　绪　论

1.1　基本知识结构图



1.2　知 识 点

1. 基本概念和术语

数据是对客观事物的符号表示，是计算机程序加工的“原料”。

数据元素是数据的基本单位，可由若干个数据项组成，数据项是数据的不可分割的最小单位。

数据对象是性质相同的数据元素的集合，是数据的一个子集。

数据结构是相互之间存在一种或多种特定关系的数据元素的集合。基本结构有4类：集

合、线性结构、树形结构、图状结构或网状结构。

存储结构即数据的物理结构，是数据结构在计算机中的表示，包括数据元素的表示和关系的表示。主要有顺序存储结构、链式存储结构、索引存储方法和散列存储方法等。

数据类型是一个值的集合和定义在这个值集上的一组操作的总称。

抽象数据类型指一个数学模型以及定义在该模型上的一组操作，仅取决于它的一组逻辑特性，而与其在计算机内部如何表示和实现无关。可通过固有数据类型来表示和实现，即利用处理器中已存在的数据类型来说明新的结构，用已实现的操作来组合新的操作。

2. 算法的定义和算法的时间、空间复杂度

算法是对特定问题求解步骤的一种描述，是指令的有限序列，其中每一条指令表示一个或多个操作。算法具有有穷性、确定性、可行性、输入和输出5个特性。“好”的算法应具有正确性、可读性、健壮性、高效率与低存储量需求。

一般情况下，算法中基本操作重复执行的次数是问题规模n的某个函数$f(n)$，算法的时间量度记作$T(n)=O(f(n))$，$T(n)$就为该算法的时间复杂度。有时要考虑平均时间复杂度和最坏时间复杂度。

空间复杂度是算法所需存储空间的量度，记作$S(n)=O(f(n))$，其中n为问题的规模。

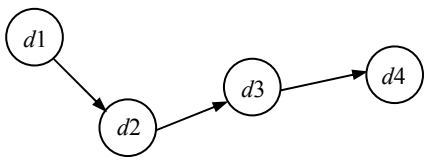
1.3 习题及参考答案

1．试描述数据结构和抽象数据类型的概念与程序设计语言中数据类型概念的区别。

【解答】简单地说，数据结构定义了一组按某些关系结合在一起的数组元素。数据类型不仅定义了一组带结构的数据元素，而且还在其上定义了一组操作。程序设计语言中的数据类型是一个值的集合和定义在这个值集上的一组操作的总称。而抽象数据类型是指一个数学模型以及定义在该模型上的一组操作。

2．设有数据结构(D, R)，其中$D=\{d1, d2, d3, d4\}$，$R=\{r\}$，$r=\{(d1, d2), (d2, d3), (d3, d4)\}$。试按图论中图的画法惯例画出其逻辑结构图。

【解答】



3．试画出与下列程序段等价的框图。

```
(1) product=1; i =1;
    while (i<=n) {
        product *=i;
        i++;
```

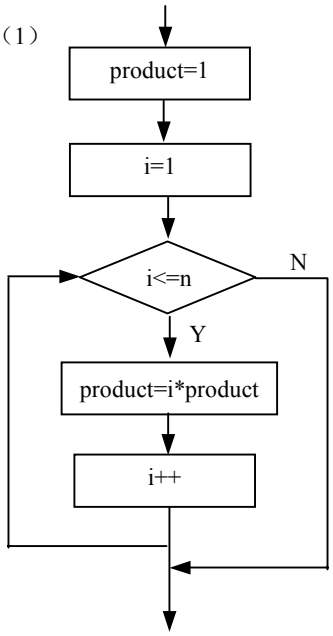
```
    }
```

（2）
```
i=0;
do{
    i++;
} while ((i!=n)&&(a[i]!=x));
```

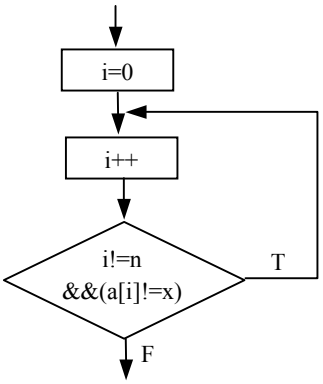
（3）
```
switch{
    case x<y:z=y-x;break;
    case x==y:z=abs(x*y);break;     //abs()为取绝对值函数
    default:z=(x-y)/abs(x)*abs(y);
}
```

【解答】

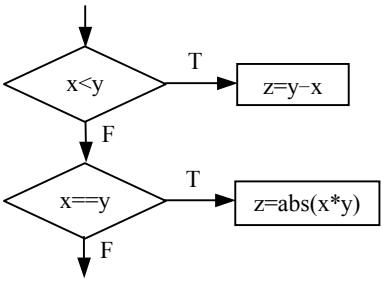
（1）



（2）



（3）



4．设*n*为正整数。试确定下列各程序段中前置以记号@的语句的频度。

（1）
```
i=1; k=0;
while (i<=n-1){
@  k+=10*i;
    i++;
}
```

（2）
```
i=1;k=0;
do {
@  k+=10*i;
    i++;
} while(i<=n-1);
```

（3）
```
i=1; k=0;
while (i<=n-1) {
```

```
        i++;
    @  k+=10*i;
    }
```

（4）
```
    k=0;
    for (i=1;i<=n;i++) {
        for (j=i;j<=n;j++)
    @     k++;
    }
```

（5）
```
    for (i=1;i<=n;i++) {
        for (j=1;j<=i;j++){
              for (k=1;k<=j;k++)
    @           x+=delta;
    }
```

（6）
```
    i=1;j=0;
    while (i+j<=n){
    @  if (i>j) j++;
        else i++;
    }
```

（7）
```
    x=n;  y=0;  //n是不小于1的常数
    while (x>=(y+1)*(y+1)) {
    @  y++;
    }
```

（8）
```
    x=91;  y=100;
    while (y>0){
    @   if (x>100) {x-=10;y--;}
         else  x++;
    }
```

【解答】（1）、（2）和（3）三个程序段中任何两段都不等效；程序段（8）取自著名的McCarthy 91函数

$$M(x)=\begin{cases} x-10 & x>100 \\ M(M(x+1)) & x\leqslant 100 \end{cases}$$

对任何$x\leqslant 100$，$M(x)=91$。此程序实质上是一个双重循环，对每个$y(>0)$值，@语句执行11次，其中10次是执行x++。

5．假设n为2的乘幂，并且$n>2$，试求下列算法的时间复杂度及变量count的值（以n的函数形式表示）。

```
int Time (int n){
    count=0;x=2;
    while(x<n/2){
        x * =2;  count++;
    }
    return(count);
} //Time
```

【解答】时间复杂度为$\log_2 n$，变量count的值为$\log_2 n-1$。

6．按增长率由小至大的顺序排列下列各函数。

2^{100}，$(3/2)^n$，$(2/3)^n$，$(4/3)^n$，n^n，$n^{3/2}$，$n^{2/3}$，$\sqrt{n}$，$n!$，n，$\log_2 n$，$n/\log_2 n$，$(\log_2 n)^2$，$\log_2(\log_2 n)$，$n\log_2 n$，$n^{\log_2 n}$

【解答】各函数的排列次序如下：

$(2/3)^n$，2^{100}，$\log_2(\log_2 n)$，$\log_2 n$，$(\log_2 n)^2$，$\sqrt{n}$，$n^{2/3}$，$n/\log_2 n$，n，$n\log_2 n$，$n^{3/2}$，$(4/3)^n$，$(3/2)^n$，$n^{\log_2 n}$，$n!$，n^n

7．已知有实现同一功能的两个算法，其时间复杂度分别为$O(2^n)$和$O(n^{10})$，假设现实计算机可连续运算的时间为10^7秒（100多天），又每秒可执行基本操作（根据这些操作来估算算法时间复杂度）10^5次。试问在此条件下，这两个算法可解问题的规模（即n值的范围）各为多少？哪个算法更适宜？

【解答】这两个算法可解问题的规模各为$n \le \log_2 10^{12}$和$n^{10} \le 10^{12}$。结论是第一个算法较适宜。由此可见，虽然一般情况下多项式阶的算法优于指数阶的算法，但高次多项式的算法在n的很大范围内不如某些指数阶的算法。

8．试设定若干n值，比较两函数n^2和$50n\log_2 n$的增长趋势，并确定n在什么范围内，函数n^2的值大于$50n\log_2 n$的值。

【解答】大约在$n>450$时，函数n^2的值才大于函数$50n\log_2 n$的值。

9．试写一算法，自大至小依次输出顺序读入的3个整数X、Y和Z的值。

【分析】假设我们仍依X、Y和Z的次序输入这3个整数，则此题的目标是做到$X \ge Y \ge Z$。在算法中应考虑对这3个元素作尽可能少的比较和移动，如下述算法在最坏的情况下只需进行3次比较和7次移动。

【解答】算法如下：

```
void Descending( ) {
    scanf (x, y, z);
    if (x<y)
        {temp=x; x=y; y=temp; }  //使x≥y
    if (y<z) {
        temp=z; z=y;           //使temp>z
        if (x≥temp) y=temp;
        else {y=x; x=temp; }
    }
    printf (x, y, z);
} //Descending
```

10．已知k阶斐波那契序列的定义为

f_0=0，f_1=0，…，f_{k-2}=0，f_{k-1}=1；

$f_n=f_{n-1}+f_{n-2}+\cdots+f_{n-k}$，$n=k$，$k+1$，…

试编写求k阶斐波那契序列的第m项值的函数算法，k和m均以值调用的形式在函数参数表中出现。

【分析】在编写此题目的函数算法的过程中，首先应根据参量m和k区分下列4种情况：（1）$m<0$；（2）$0\leq m<k-1$；（3）$m=k-1$；（4）$m\geq k$。其次在计算$m\geq k$的f_m值时，可先对计算公式作数学处理，将f_{i+1}表示为f_i和f_{i-k}的简单函数；最后考虑计算f_n所需的辅助空间。

【解答】算法如下：

```
Status fib(int k, int m, int &f)//求k阶斐波那契序列的第m项的值f
{
    int tempd;
    if(k<2 || m<0) return ERROR;
    if(m<k-1) f=0;
    else if (m==k-1) f=1;
        else
        {
            for(i=0;i<=k-2;i++) temp[i]=0;
            temp[k-1]=1; //初始化
            for(i=k;i<=m;i++) //求出序列第k至第m个元素的值
            {
               sum=0;
               for(j=i-k;j<i;j++) sum+=temp[j];
               temp[i]=sum;
            }
            f=temp[m];
         }
      return OK;
}//fib
```

11．假设有A，B，C，D，E 5个高等院校进行田径对抗赛，各院校的单项成绩均已存入计算机，并构成一张表，表中每一行的形式为：

项目名称	性别	校名	成绩	得分

试编写算法，处理上述表格，以统计各院校的男、女总分和团体总分，并输出。

【分析】设置此题目的目的在于复习结构型变量的使用方法，在此题中可设

```
typedef struct{
    char *sport;
    enum{male, female} gender;
    char schoolname; //校名为'A', 'B', 'C', 'D'或'E'
    char *result;
    int score;
} resulttype;
typedef struct{
    int malescore;
```

```
    int femalescore;
    int totalscore;
} scoretype;
```

【解答】算法如下：

```
void summary(resulttype result[ ])//求各校的男女总分和团体总分，假设结果已经储存在result[ ]数组中
{
    scoretype score ;
    i=0;
    while(result[i].sport!=NULL)
    {
        switch(result[i].schoolname)
        {
            case 'A':
                score[0].totalscore+=result[i].score;
                if(result[i].gender==0) score[0].malescore+=result[i].score;
                else score[0].femalescore+=result[i].score;
                break;
            case 'B':
                score [1].totalscore+=result[i].score;
                if(result[i].gender==0) score[1] .malescore+=result[i].score;
                else score[1] .femalescore+=result[i].score;
                break;
                ……
            }
            i++;
        }
        for(i=0;i<5;i++)
    {
        printf("School %d:\n", i);
        printf("Total score of male:%d\n", score[i].malescore);
        printf("Total score of female:%d\n", score[i].femalescore);
        printf("Total score of all:%d\n\n", score[i].totalscore);
    }
}//summary
```

12. 试编写算法，计算$i!\cdot 2^i$(i=0，1，…，n-1)的值并分别存入数组a[arrsize]的各个分量中。假设计算机中允许的整数最大值为MAXINT，则当n>arrsize或对某个k($0\leqslant k\leqslant n-1$)使$k!\cdot 2k$>MAXINT时，应按出错处理。注意选择你认为较好的出错处理方法。

【分析】注意MAXINT为计算机中允许出现的最大值，则在过程体中不能以计算所得结果大于MAXINT作为判断出错的依据。当某一项的结果超过了MAXINT时，它除以前面一项的商会发生异常。

【解答】算法如下：

```
Status algo119(int a[arrsize])//求i!*2^i序列的值且不超过MAXINT
{
    last=1;
```

```
    for(i=1;i<=arrsize;i++)
    {
        a[i-1]=last*2*i;
        if((a[i-1]/last)!=(2*i)) return OVERFLOW;
        last=a[i-1];
        last=a[i-1];
        return OK;
    }
}//algo119
```

13．试编写算法求一元多项式$P_n(x)=\sum_{i=0}^{n}a_i x^i$的值$P_n(x_0)$，并确定算法中每一语句的执行次数和整个算法的时间复杂度。注意选择你认为较好的输入和输出方法。本题的输入为$a_i(i=0, 1, \cdots, n)$、x_0和n，输出为$P_n(x_0)$。

【分析】注意计算过程中，不要对多项式中的每一项独立计算x的幂。

【解答】算法如下：

```
void polyvalue()
{
    float ad;
    float *p=a;
    printf("Input number of terms:");
    scanf("%d", &n);
    printf("Input the %d coefficients from a0 to a%d:\n", n, n);
    for(i=0;i<=n;i++) scanf("%f", p++);
    printf("Input value of x:");
    scanf("%f", &x);
    p=a;xp=1;sum=0; //xp用于存放x的i次方
    for(i=0;i<=n;i++)
    {
        sum+=xp*(*p++);
        xp*=x;
    }
    printf("Value is:%f", sum);
}//polyvalue
```


1.4 考研真题分析

例题1-1 （中国科学院软件研究所1999年试题）

判断正误：

① 顺序存储方式只能用于存储线性结构。

② 顺序查找法适用于存储结构为顺序或链接存储的线性表。

【解答】① 错误。顺序存储方式也可以用来存储树型结构，例如堆排序中的堆。

② 正确。

例题1-2 （中国科学院计算机网络信息中心2001年试题）

单项选择题：

计算机算法必须具备输入、输出、_____等5个特性。

A．可行性、可移植性和可扩展性　　B．可行性、确定性和有穷性

C．确定性、有穷性和稳定性　　D．易读性、安全性和稳定性

【解答】B

例题1-3 （中国科学院计算机网络信息中心2001年试题）

单项选择题：

设三个函数f，g，h分别为

$f(n)=100n^3+n^2+1000$

$g(n)=25n^3+4000n^2$

$h(n)=n^{1.01}+1000n\log_2 n$

以下关系式中有错的是_____。

A．$f(n)=O(g(n))$　　B．$g(n)=O(f(n))$

C．$h(n)=O(n^{1.01})$　　D．$h(n)=O(n\log_2 n)$

【解答】D

例题1-4 （中国科学院计算机网络信息中心2001年试题）

问答题：

已知在串匹配的KMP算法中求Next函数的算法为

```
void get_next(char *t, int *next){
    for(j=1, k=0, next[1]=0;j<strlen(t);)
       if((k==0)  ||  (t[j]==t[k]) ) {
          j++, k++;
          next[j]=k;
       }
       else k=next[k];
}/ / end get_next
```

试给出该函数的时间复杂度并证明该结论。

【解答】设strlen(t)=m，则时间复杂度为$O(m)$。

【证明】考虑算法中j-k的值，在算法执行过程中j-k值只增不减，而j-k增加的次数小于等于j的最大值m，j-k得数不变的次数小于等于j增加的次数m，而算法中要么j增加，要么j-k增加，故总执行次数小于等于2m，因此，算法的时间复杂度为$O(m)$。

例题1-5 （中国科学院计算机网络信息中心2001年试题）

算法题：

设L是长度为n的单链表的头指针，不带头结点，每个表结点中存放一个字符。请写一算法判断L中存放的字符串是否是中心对称的，要求附加的空间是$O(1)$，时间复杂度为$O(n)$。

【分析】① 先计算出n值；
② 再将前$n/2$个结点作为一个链表，后$n/2$个结点也作为一个链表并逆置；
③ 然后依次判断两个单链表对应元素是否相同。
应注意边量条件处理，即n为奇数的情况。

【解答】算法如下：

```
PROC checkstr(var L:linkisttp; n:integer);
VAR p, h, q:linkisttp;
     i:integer;
BEGIN
    P:=L;
    if (n mod z =0) then
        for i:=1 to (n/2-1)do
           p:=p↑.next;
    else
        for i:=1 to n-1/2 do
           p:=p↑.next;
    h:=p↑.next;
    p↑.next:=nil;
    p:=h↑.next;
    h↑.next:=nil;
    q:=p;
    while q<>nil do
    begin
        q:=p↑.next;
        p↑.next:=h;
        h:=p;
        p:=q;
    end
    p:=L;
    q:=h;
    while q<> nil do
        if (p↑.data<>q↑.data) then
        return (false)
        else begin
           p:=p↑.next;
           q:=q↑.next;
        end
    return (true);
ENDP
```

例题1-6 （中国科学院计算机网络信息中心2000年试题）

下述函数中渐近时间最小的是_____。

A. $T_1(n)=n\log_2 n+1000\log_2 n$　　B. $T_2(n)=n^{\log_2^3}-1000\log_2 n$

C. $T_3(n)=n^2-1000\log_2 n$　　D. $T_4(n)=2n\log_2 n-1000\log_2 n$

【解答】A

例题1-7 （北京科技大学2002年试题）

数据的逻辑结构在计算机存储器中的映像（或表示）通常有哪几种方法？

【解答】顺序映像和非顺序映像

例题1-8 （北京科技大学2002年试题）

请简述算法的确定性之含义。

【解答】算法的确定性是指算法中每一条指令必须有确切的含义，读这些指令时不会产生二义性；并且在任何条件下，算法只有惟一的一条执行路径，即对于相同的输入只能得到相同的输出。

例题1-9 （北京科技大学2002年试题）

线性结构和树形结构的特点分别是什么？

【解答】
线性结构：结构中的数据元素之间存在一个对一个的关系。
树形结构：结构中的数据元素之间存在一个对多个的关系。

例题1-10 （武汉理工大学2002年试题）

选择题：
算法在发生非法操作时可以作出处理的特性称为_____。
A．正确性　　B．易读性　　C．健壮性　　D．可靠性

【解答】C

例题1-11 （武汉理工大学2002年试题）

简答题：
简述顺序存储结构与链式存储结构在表示数据元素之间关系上的主要区别。

【解答】在顺序存储结构中，逻辑关系上相邻的两个元素在物理位置上也相邻，可以随机存取表中任一元素；而链式存储结构中，数据元素之间的关系是由结点中的指针指示的。

例题1-12 （南京理工大学2002年试题）

简答题：
简述算法的5个特性。

【解答】（1）有穷性：一个算法必须总是在执行有穷步之后结束，且每一步在有穷时间内完成；（2）确定性：每条指令有确切的含义，且在任何条件下，算法只有惟一的一条执行路径；（3）可行性：算法中描述的操作都是可以通过已实现的基本运算执行有限次来实现；（4）输入：一个算法有零个或多个输入；（5）输出：一个算法有一个或多个输出。

例题1-13 （南京理工大学2002年试题）

选择题：

数据结构是一门研究非数值计算的程序设计问题中计算机的（1）以及它们之间的(2)和运算的学科。

（1）A．操作对象B．计算方法　　C．逻辑存储　　D．数据映像

（2）A．结构　　B．关系　　C．运算　　D．算法

【解答】（1）A　　（2）B

例题1-14 （南京理工大学2002年试题）

选择题：

在数据结构中，逻辑上数据结构可分为_____。

A．动态结构和静态结构　　B．线性结构和非线性结构

C．紧凑结构和非紧凑结构　　D．内部结构和外部结构

【解答】B

例题1-15 （南京理工大学2002年试题）

写出对顺序表作直接插入排序的算法。

【解答】算法如下：

```
void InsertSoft(ElemType A[ ], int n)
{
    ElemType x;
    int i, j;
    for (i=1;i<n;i++)  //i表示插入次数；共进行n-1次插入
    {  x=A[i];
         for(j=i-1;j>=0;j--)
             if(x.stn<A[j].stn)
                 A[j+1]=A[j]  //进行恢复比较和插入
             else
                 break;      //查询到j+1位置时离开j循环
        A[j+1]=x;   //把原A[i]的值插入到下标为j+1的位置
    }
}
```

例题1-16 （武汉大学2002年试题）

名词解释：

（1）数据对象　　（2）物理结构　　（3）空间复杂度

【解答】（1）数据对象：简称对象，它属于一种数据类型（包括一般数据类型和抽象数据类型）中的特定量（又称实例），包括变量和常量。

（2）物理结构：又称为存储结构，是指一种数据结构在存储器中的存储方式。数据的存储方式有4种，即顺序存储方式、链式存储方式、索引存储方式和散列存储方式。

（3）空间复杂度：是对一个算法在运行过程中临时占用存储空间大小的量度。一个算法在计算机存储器上所占用的存储空间，包括存储算法本身所占用的存储空间，算法的输入输出数据所占用的存储空间和算法在运行过程中临时占用的存储空间3个方面。

1.5 自 测 题

自测题1-1

在下面的函数*P*中，参数*x*是通过地址传递的，参数*y*通过值传递：

```
FUNCTION  P(VAR x:integer; y:integer):integer;
BEGIN k:=3;
    l:=5;
    P:=x+y
END;
```

若*P*由下列程序调用：

```
k:=1;
l:=1;
z:=P(k, l)
```

则问*z*的值是什么？这两种参数传递机制如何实现？

自测题1-2

下面是一个FORTRAN 77程序。按语言的语义，程序的输出结果是什么？在静态存储分配情况下，实际的输出结果是什么？两者是否有区别？试说明理由。

```
CALL SUB
CALL SUB
END

SUBROUTINE SUB
DATA I/10/
WRITE (*, *) I
I=100
END
```

自测题1-3

下面是一个C语言程序及其运行结果。从运行结果看，函数FUNC中4个局部变量*i*1、*j*1、*f*1和*e*1的地址间隔和它们类型的大小是一致的；而4个形式参数*i*、*j*、*f*和*e*的地址间隔和它们类型的大小不一致，试分析不一致的原因：

```
#include<stdio.h>
FUNC(i, j, f, e)
short i, j;  float f, e;
{
```

```
    short i1, j1;    float f1, e1;
    i1=i;  j1=j;  f1=f;  e1=e;
    printf("Addresses of i, j, f, e=%d1  %d1  %d1  %d1\n", &i, &j, &f, &e);
    printf("Addresses of i1, j1, f1, e1=%d1 %d1 %d1
        %d1\n", &i1, &j1, &f1, &e1);
    printf("Sizes of short, int, long, float, double=%d1 %d1 %d1 %d1 %d1\n")
        sizeof(short), sizeof(int), sizeof(long), sizeof(float), sizeof (double));
}
main( ){
    short i, j;float f, e;
    i=j=1;f=e=1.0;
    FUNC(i, j, f, e);
}
```

运行结果为:

```
Addresses of i, j, f, e=-268438178, -268438174, -268438172, -268438164
Addresses of i1, j1, f1, e1=-268438250, -268438252, -26843256, -26843260
Sizes of short,  int,  long,  float,  double=2, 4, 4, 4, 8
```

自测题1-4

将下列函数，按它们在$n\to\infty$时的无穷大阶数，从小到大排列：

n，$n-n^3+7n^5$，$n\log n$，$2^{n/2}$，n^3，$\log n$，$n^{1/2}+\log n$，$(3/2)^n$，$n!$，$n^2+\log n$

1.6 自测题答案

【自测题1-1】

这是一类题型，必须清楚传名、传地址、传结果以及传值这4类参数机制的区别，才不至于混淆。

当传值时，在调用段对参数进行的修改不会影响主程序段中的参数实际值；而传地址时，在调用段对参数的修改就改变了实际参数值。

因为x是传地址，在调用z=P(k，l)时，k的地址就传到调用段中，而y为传值，所以l只有值传过去了，在调用P中，k:=3，故k的实际值也就是x的值，都同时变为3，而y只是l的值，对l:=5的修改并不能改变y的值，故此时x=3，y仍为1，所以P:=x+y，即为P:=3+1=4，故返回为P:=4，即z=4。

【扩展】注意，除了传值与传地址两种方法外，还有“传结果”与“传名”两种传递参数的机制。

“传结果”机制是：每个形式参数对应两个单元，第1个单元存放实参的地址，第2个单元存放实参的值。在过程体中对形参的任何引用或赋值都被看成是对它的第2个单元的直接访问，但过程完毕返回前必须把第2个单元的内容存放到第1个单元所指实参单元中。

“传名”机制是：过程调用的作用相当于把被调用段的过程体抄到调用出现的地方，

但把其中任一出现的形参都替换为相应的实参（文字替换）。这种替换方式的基本实现方法是：在进入被调用段之前不对实参预先进行计算，而是让过程体中每当使用到相当的形参时才逐次对它进行计算。因此，在实现时，通常都把实参处理成一个子程序，每当过程体中使用到相应形参时就调用这个子程序。

【自测题1-2】

按FORTRAN 77的语义，子程序SUB的第1次执行输出10，第2次输出的值不确定。FORTRAN语言的实现一般都采用静态存储分配，变量和存储单元的结合（Binding）是静态决定的，在程序运行期间不会改变。所以如果采取静态存储分配，所有对I的操作都是对同一实际地址空间的操作，第1次调用SUB和第2次进入时，都对这个地址操作，因此SUB的两次执行的输出分别是10、100。

【扩展】对于动态存储与静态存储的概念一定要区别清楚。在子程序的调用，特别是递归调用中，这两者会有本质的不同。静态存储是对一个变量固定只分配一个地址，以后不管在哪一层中对此变量进行改动，都是在对此地址中的实际对象做的改动，都是有效的；而动态存储并没有对一个变量固定分配一个地址空间，而是动态地不断为其分配不同的空间，在对应子程序执行完毕后又回收其子空间，此时在此空间中对变量进行的改动自然也不会保留下来了。因此在动态存储中，一层对变量进行的改动往往不会对另一层的变量造成影响，每次定义此变量时，变量都得到了“新生”。这就是动态存储与静态存储的本质区别。

【自测题1-3】

C编译是不作调用时形参和实参一致性检查的。注意在C语言中，整数有short、int和long共3种类型，实数有float和double两种类型。C语言的参数调用是按传值方式进行的，一个整型表达式的值究竟是按short、int，还是按long方式传递呢？

显然，这几约定为应该按取数的最大方式来读取参数，即对于整数用long方式，实数用double方式。虽然参数传递是按最大方式进行的，但是被调用函数中形式参数是按自己的类型定义来取值的。这样一来，参数传递和取值是不一致的，因此就要考虑边界对齐问题。因float类型为4字节，而double类型为8字节，故当从double类型变量取float类型的值时，应从第1~4个字节分配float类型数。short型的形参是取long型值4字节中的后两字节内容，float型的形参是取double值8字节的前4字节，这样才出现这4个形参地址与它们的类型大小不一致的结果。

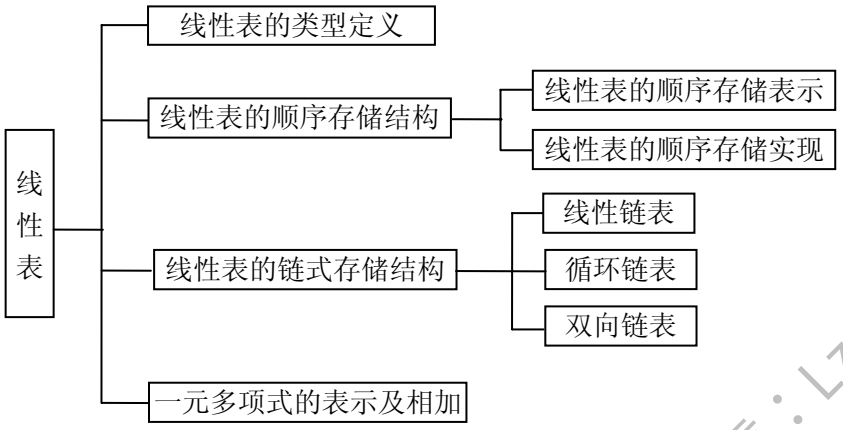
【自测题1-4】

从小到大排列如下：

$\log n$， $n^{1/2}+\log n$， n， $n\log n$， $n^2+\log n$， $n3$， $n-n^3+7n^5$， $2^{n/2}$， $(3/2)^n$， $n!$

第 2 章　线 性 表

2.1　基本知识结构图



2.2　知 识 点

1. 线性表的类型定义

线性表是一种线性结构，它具有线性结构的特点，即在数据元素的非空有限集中：

（1）存在惟一一个被称作“第一个”的数据元素；
（2）存在惟一一个被称作“最后一个”的数据元素；
（3）除第一个之外，集合中的每个数据元素均只有一个前驱；
（4）除最后一个之外，集合中的每个数据元素均只有一个后继。

一个线性表是n个数据元素的有限序列。每个数据元素可以是一个数或一个符号，甚至是若干个数据项组成的记录等。

2. 定义在线性表上的操作

可以访问线性表中的数据元素，也可以对数据元素进行查找、插入、删除等操作，或者求某数据元素的前驱、后继元素等。

3. 线性表的顺序和链式存储结构

线性表的顺序存储结构是指使用一组地址连续的存储单元依次存储线性表中的数据元素。

线性表的链式存储结构是指用一组任意的存储单元存储线性表的数据元素。对每一个数据元素除了存储其本身的信息（数据域）外，还需存储一个指示其直接后继的信息（指针域）。链式存储结构有3种形式：线性链表、循环链表和双向链表。

在顺序存储结构中，逻辑上相邻的两个元素在物理位置上也相邻，因此可以随机存取表中任意元素；而在链式存储结构中，虽然不能对数据元素随机存取，但却换来了对存储空间操作的方便性，而且在插入或删除操作时不需移动大量元素。

2.3　习题及参考答案

1．描述以下3个概念的区别：头指针，头结点，首元结点（第一个元素结点）。

【解答】首元结点是指链表中存储线性表中第一个数据元素a_1的结点。为了操作方便，通常在链表的首元结点之前附设一个结点，称为头结点，该结点的数据域中不存储线性表的数据元素，其作用是当对链表进行操作时，可以对空表、非空表的情况以及对首元结点进行统一处理。头指针是指向链表中第一个结点（或为头结点或为首元结点）的指针。若链表中附设头结点，则不管线性表是否为空表，头指针均不为空，否则表示空表的链表的头指针为空。这3个概念对单链表、双向链表和循环链表均适用。是否设置头结点，是不同的存储结构表示同一逻辑结构的问题。

2．填空题。

（1）在顺序表中插入或删除一个元素，需要平均移动_____元素，具体移动的元素个数与_____有关。

（2）顺序表中逻辑上相邻的元素的物理位置_____紧邻。单链表中逻辑上相邻的元素的物理位置_____紧邻。

（3）在单链表中，除了首元结点外，任一结点的存储位置由_____指示。

（4）在单链表中设置头结点的作用是_____。

【解答】（1）表中一半；表长和该元素在表中的位置

（2）必定；不一定

（3）其直接前驱结点的链域的值

（4）插入和删除首元素时不必进行特殊处理

3．在什么情况下用顺序表比链表好？

【解答】操作过程中不需要移动大量数据时用顺序表比链表好。

4．已知*L*是无表头结点的单链表，且P结点既不是首元结点，也不是尾元结点，试从下列提供的答案中选择合适的语句序列。

a. 在P结点后插入S结点的语句序列是_____。

b. 在P结点前插入S结点的语句序列是_____。

c. 在表首插入S结点的语句序列是_____。

d. 在表尾插入S结点的语句序列是_____。

（1）P->next=S;

（2）P->next=P->next->next;

（3）P->next=S->next;

（4）S->next=P->next;

（5）S->next=L;

（6）S->next=NULL;

（7）Q=P;

（8）while (P->next!=Q) P=P->next;

（9）while (P->next!=NULL) P=P->next;

（10）P=Q;

（11）P=L;

（12）L=S;

（13）L=P;

【解答】a.（4）（1） b.（7）（11）（8）（4）（1） c.（5）（12）d.（9）（1）（6）

5. 已知*L*是带表头结点的非空单链表，且P结点既不是首元结点，也不是尾元结点，试从下列提供的答案中选择合适的语句序列。

a. 删除P结点的直接后继结点的语句序列是_____。

b. 删除P结点的直接前驱结点的语句序列是_____。

c. 删除P结点的语句序列是_____。

d. 删除首元结点的语句序列是_____。

e. 删除尾元结点的语句序列是_____。

（1）P=P->next;

（2）P->next=P;

（3）P->next=P->next->next;

（4）P=P->next->next;

（5）while (P!=NULL) P=P->next;

（6）while (Q->next!=NULL) {P=Q; Q=Q->next;}

（7）while (P->next!=Q) P=P->next;

（8）while (P->next->next!=Q) P=P->next;

（9）while (P->next->next!=NULL) P=P->next;

（10）Q=P;

（11）Q=P->next;

（12）P=L;
（13）L=L->next;
（14）free(Q);

【解答】a.（3）（1）（14） b.（10）（12）（8）（11）（3）（14） c.（10）（12）（7）（3）（14） d.（12）（3）（11）（14） e.（9）（11）（3）（14）

6. 已知P结点是某双向链表的中间结点，试从下列提供的答案中选择合适的语句序列。

a. 在P结点后插入S结点的语句序列是_____。
b. 在P结点前插入S结点的语句序列是_____。
c. 删除P结点的直接后继结点的语句序列是_____。
d. 删除P结点的直接前驱结点的语句序列是_____。
e. 删除P结点的语句序列是_____。

（1）P->next=P->next->next;
（2）P->prior=P-> prior -> prior;
（3）P->next=S;
（4）P->prior=S;
（5）S->next=P;
（6）S->prior=P;
（7）S->next=P->next;
（8）S->prior=P-> prior;
（9）P->prior->next=P->next;
（10）P->prior->next=P;
（11）P->next-> prior =P;
（12）P->next-> prior =S;
（13）P-> prior ->next=S;
（14）P->next-> prior =P-> prior;
（15）Q=P->next;
（16）Q=P-> prior;
（17）free(P);
（18）free(Q);

【解答】a.（7）（12）（3）（6） b.（8）（13）（5）（4） c.（15）（1）（11）（18） d.（16）（2）（10）（18） e.（9）（14）（17）

7. 简述以下算法的功能。

（1）
```
Status A(LinkedList L)   {  //L是无表头结点的单链表
        if (L && L->next) {
          Q=L; L=L->next; P=L;
          while (P->next) P=P->next;
          P->next=Q; Q->next=NULL;
```

```
        }
        return OK;
    } //A
```

（2）

```
void BB (LNode *s,  LNode *q ) {
        p=s;
        while (p->next!=q) p=p->next;
        p->next=s;
    } //BB
    void AA(LNode *pa,  LNode *pb) {
        // pa和pb分别指向单循环链表中的两个结点
        BB(pa, pb);
        BB(pb, pa);
    } //AA
```

【解答】（1）如果*L*的长度不小于2，则将首元结点删去并插入到表尾。

（2）将s至q之前的结点组成一个新的单循环链表，将q断开。

8．指出以下算法中的错误和低效（即费时）之处，并将它改写为一个既正确又高效的算法。

```
Status DeleteK(SqList&a, int i, int k) {
    //本过程从顺序存储结构的线性表a中删除第i个元素起的k个元素
    if (i<1  ||  k<0  ||  i+k>a. length) return INFEASIBLE;  //参数不合法
    else{
        for (count=1; count<k; count++) {
        //删除一个元素
            for (j=a. length; j>=i+1; j--) a.elem[j-1]=a.elem[j];
            a.length--; }
    }
    return OK;
} //DeleteK
```

【解答】错误有两处：

（1）参数不合法的判别条件不完整。合法的入口参数条件为

（0<i≤a.length）∧（0≤k≤a.length -i）

（2）第二个for语句中，元素前移的次序错误。

低效之处是每次删除一个元素的策略。

9．设顺序表*va*中的数据元素递增有序。试写一算法，将x插入到顺序表的适当位置上，以保持该表的有序性。

【分析】此题的算法思想：

（1）查找x在顺序表a.elem[a.length]中的插入位置，即求满足a.elem[i]≤x<a.elem[i+1]的i值（i的初值为a.length -1）；

（2）将顺序表中的a.length -i -1个元素a.elem[i+1..a.length-1]后移一个位置；

（3）将x插入到a.elem[i+1]中且将表长a.length加1。

上述算法正确执行的参数条件为：0≤a.length<a.listsize。

【解答】算法如下：

```
Status InsertOrderList (SqList &a,  ElemType x)
{
    //顺序表a中的元素依值递增有序，本算法将x插入其中适当位置
    //以保持其有序性。入口断言：0<=a.length<a.listsize
    if (a.length = =a.listsize) return (OVERFLOW);
    else {
      i= a.length-1;
      while (i>=0 && x<a.elem[i] ) i--;    //查找x的插入位置
      // i<0 ∨ x >= va.elem[i]
      for (j=a.length-1; j>=i+1; j--)
        a.elem[j+1]=a.elem[j];             //元素后移
      a.elem[i+1]=x;                       //插入x
      a.length++;                          //表长加1
      return OK;
    }
} //InsertOrderList
```


注意

（1）算法中设置的断言表明以下程序代码正确执行时所要求满足的参数条件，或表明以上程序代码执行后所达到的变量状态；

（2）在while循环中，条件与&&采用类C的定义，其作用是避免当i<0时发生数组a.elem越界的错误；

（3）注意上述算法在a.length=0时也能正确执行；

（4）可以将上述算法中元素后移的动作并入查找的while循环中一起完成。即删去上述算法中的for 循环语句，且将while循环语句改为下列形式：

```
while ( i >= 0 && x<a.elem[i] )
{a.elem[i+1]=va.elem[i]; i--; }
```

10．试写一算法在带头结点的单链表结构上实现线性表操作LOCATE(L，X)。

【分析】10~12题是对单链表的一些基本操作，要注意提前保留结点指针，以备在后序过程中用到，尤其是在链表的合并、结点的删除中。

【解答】算法如下：

```
LNode* Locate(LinkList L，int x)//链表上的元素查找，返回指针
{
  for(p=L->next;p&&p->data!=x;p=p->next);
  return p;
}//Locate
```

11．试写一算法在带头结点的单链表结构上实现线性表操作LENGTH(L)。

【解答】算法如下：

```
int Length(LinkList L)//求链表的长度
{
```

```
    for(k=0, p=L;p->next;p=p->next, k++);
    return k;
}//Length
```

12．已知指针ha和hb分别指向两个单链表的头结点，并且已知两个链表的长度分别为m和n，试写一算法将这两个链表连接在一起（即令其中一个表的首元结点连在另一个表的最后一个结点之后）。假设指针hc指向连接后的链表的头结点，并要求算法以尽可能短的时间完成连接运算。请分析你的算法的时间复杂度。

【解答】算法如下：

```
void ListConcat(LinkList ha, LinkList hb, LinkList &hc)//把链表hb接在ha后面形成链表hc
{
    hc=ha;
    p=ha;
    while(p->next) p=p->next;
    p->next=hb;
}//ListConcat
```

该算法的时间复杂度为$O(m+n)$。

13．已知指针la和lb分别指向两个无头结点单链表中的首元结点。下列算法是从表la中删除自第i个元素起共len个元素后，将它们插入到表lb中第j个元素之前。试问此算法是否正确？若有错，则请改正之。

```
Status DeleteAndInsertSub (LinkedList la, LinkedList lb, int i, int j, int len){
      if (i<0 || j<0 || len<0)return INFEASIBLE;
      p=la; k=1;
      while(k< i){p=p->next; k++;}
      q=p;
      while (k<=len){q=q->next;k++;}
      s=lb;k=1;
      while(k<j){s=s->next; k++;}
      s->next=p; q->next=s->next;
      return OK;
}//DeleteAndInsertSub
```

【分析】注意此题中的条件是，采用的存储结构（单链表）中无头结点，因此在写算法时，特别要注意空表和第一个结点的处理。算法中尚有其他类型的错误，如结点的计数，修改指针的次序等。

【解答】正确算法如下：

```
Status DeleteAndInsertSub (LinkedList &la,  LinkedList &lb,  int i,  int j,  int len )
{
    //la和lb分别指向两个单链表中第一个结点，本算法是从la表中删去第i个
    //元素起共len个元素，并将它们插入到lb表中第j个元素之前，若lb表中只
    //有j-1个元素，则插在表尾。
    //入口断言：(i>0)∧(j>0)∧(len>0)
    if (i<0  ||  j<0  ||  len<0) return INFEASIBLE;
    p=la; k=1; prev=NULL;
```

```
    while (p &&k<i )    //在la表中查找第i个结点
    {
        prev=p;  p=p->next;  k++;
    }
    if (!p ) return INFEASIBLE;
    q=p;  k=1;          //p指向la表中第i个结点
    while (q &&k<len) {q=q->next;  k++; }  //查找la表中第i+len-1个结点
    if (!q) return INFEASIBLE;
    if (!prep)  la=q->next;    //i=1的情况
    else  prep->next=q->next;    //完成删除
    //将从la中删除的结点插入到lb中
    if (j ==1) {q->next=lb;  lb=p; }
    else {  // j>=2
          s=lb;  k=1;
          while (s &&k<j-1)  {s=s->next;  k++; }
          //查找lb表中第j-1个元素
          if (!s) return INFEASIBLE;
          q->next=s->next;  s->next=p;   //完成插入
          return OK;
          }
} //DeleteAndInsertSub
```

14. 试写一算法，在无头结点的动态单链表上实现线性表操作INSERT(L，i，b)，并和在带头结点的动态单链表上实现相同操作的算法进行比较。

【解答】算法如下：

```
Status Insert(LinkList &L，int i，int b) //在无头结点链表L的第i个元素之前插入元素b
{
    p=L;
    q=(LinkList*)malloc(sizeof(LNode));
    q.data=b;
    if(i==1)
    {
        q.next=p;L=q; //插入在链表头部
    }
    else
    {
        while(--i>1) p=p->next;
        q->next=p->next;p->next=q; //插入在第i个元素的位置
    }
}//Insert
```

15. 同14题要求，试写一算法，实现线性表操作DELETE(L，i)。

【解答】算法如下：

```
Status Delete(LinkList &L，int i)//在无头结点链表L中删除第i个元素
{
    if(i==1) L=L->next; //删除第一个元素
    else
    {
```

```
        p=L;
        while(--i>1) p=p->next;
        p->next=p->next->next; //删除第i个元素
    }
}//Delete
```

16. 已知线性表中的元素以值递增有序排列，并以单链表作为存储结构。试写一高效的算法，删除表中所有值大于mink且小于maxk的元素（若表中存在这样的元素）同时释放被删结点空间，并分析你的算法的时间复杂度（注意：mink和maxk是给定的两个参变量，它们的值可以和表中的元素相同，也可以不同）。

【分析】合法的入口条件只要求线性表不空，若mink<maxk，则表明待删元素集为空集。注意题中要求的“高效”算法指的是，应利用“元素以值递增有序排列”的已知条件，被删元素集必定是线性表中连续的一个元素序列。则在找到第1个被删元素时，应保存指向其前驱结点的指针。注意在删除结点的同时释放它的空间。

【解答】算法如下：

```
Status Delete_Between(Linklist &L, int mink, int maxk)//删除元素递增排列的链表L中值大于mink
                                                      //且小于maxk的所有元素
{
    p=L;
    while(p->next->data<=mink) p=p->next; //p是最后一个不大于mink的元素
    if(p->next)    //如果还有比mink更大的元素
    {
        q=p->next;
        while(q->data<maxk) {free(q);q=q->next;} //q是第1个不小于maxk的元素
        p->next=q;
    }
}//Delete_Between
```

该算法的时间复杂度为$O(n)$。

17. 同16题条件，试写一高效的算法，删除表中所有值相同的多余元素（使得操作后的线性表中所有元素的值均不相同），同时释放被删结点的空间，并分析你的算法的时间复杂度。

【分析】类似16题解法，只需记录相等元素部分的头尾，再改变指针指向即可。

【解答】算法如下：

```
Status Delete_Equal(Linklist &L)      //删除元素递增排列的链表L中所有值相同的元素
{
    p=L->next;q=p->next;              //p，q指向相邻两元素
    while(p->next)
    {
        if(p->data!=q->data)
        {
            p=p->next;q=p->next;      //当相邻两元素不相等时，p，q都向后推一步
        }
```

```
        else
        {
            while(q->data==p->data)
            {
                free(q);
                q=q->next;
             }
             p->next=q;p=q;q=p->next; //当相邻元素相等时删除多余元素
        }//else
    }//while
}//Delete_Equal
```

该算法的时间复杂度为$O(n-1)$。

18. 试写一算法，实现顺序表的就地逆置，即利用原表的存储空间将线性表（a_1，a_2，…，a_n）逆置为（a_n，a_{n-1}，…，a_1）。

【分析】理解就地逆置的含义，即仍要利用原有存储空间，设置一个临时变量q，再从两个方向进行表头表尾的交换。

【解答】算法如下：

```
void reverse(SqList &A)//顺序表的就地逆置
{
     int q;
     for(i=1, j=A.length;i<j;i++, j--)
          q=A.elem[i];
     A.elem[i]=A.elem[j];
     A.elem[j]=q;
}//reverse
```

19. 试写一算法，对单链表实现就地逆置。

【分析】以单链表作为存储结构进行就地逆置的正确做法应该是：将原链表中的头结点和第一个元素结点断开（令其指针域为空），先构成一个新的空表，然后从第一个结点起，将原链表中各结点依次插入到这个新表的头部（即令插入的结点成为新的第一个元素结点）。

【解答】算法如下：

```
void LinkList_reverse(Linklist &L)//链表的就地逆置；为简化算法，假设表长大于2
{
    p=L->next;q=p->next;s=q->next;p->next=NULL;
    while(s->next)
    {
        q->next=p;p=q;
        q=s;s=s->next; //把L的元素逐个插入新表表头
    }
    q->next=p;s->next=q;L->next=s;
}//LinkList_reverse
```

20. 设线性表$A=(a_1, a_2, \cdots, a_n)$，$B=(b_1, b_2, \cdots, b_n)$，试写一个按下列规则合并$A$，$B$

为线性表C的算法，即使得

$C=(a_1, b_1, \cdots, a_m, b_m, b_{m+1}, \cdots, b_n)$　　当$m \leqslant n$时

或者　　$C=(a_1, b_1, \cdots, a_n, b_n, a_{n+1}, \cdots, a_m)$　　当$m > n$时

线性表A、B和C均以单链表作为存储结构，且C表利用A表和B表中的结点空间构成。

注意：单链表的长度值m和n均未显式存储。

【分析】假设以表A的头结点作为表C的头结点，则自a_1和b_1起，交替将表A和表B中的结点链接到表C上去。假设指针pc指向新的表C中当前最后一个结点，pa和pb分别指向表A和表B中当前尚未链接到表C上的（剩余部分）第一个结点，则pc->next域或者指向pa所指结点，或者指向pb所指结点，使每次循环在表C中只增加1个结点。注意循环进行的条件是什么？跳出循环后还应进行什么操作？还应注意最后释放表B的头结点。

【解答】算法如下：

```
void merge1(LinkList &A, LinkList &B, LinkList &C)//把链表A和B合并为C，A和B的元素间隔排列，
                                                   // 且使用原存储空间
{
    p=A->next;q=B->next;C=A;
    while(p&&q)
    {
        s=p->next;p->next=q; //将B的元素插入
        if(s)
        {
           t=q->next;q->next=s; //如A非空，将A的元素插入
        }
        p=s;q=t;
    }//while
}//merge1
```

21．假设有两个按元素值递增有序排列的线性表A和B，均以单链表作存储结构，请编写算法将A表和B表归并成一个按元素值递减有序（即非递增有序，允许表中含有值相同的元素）排列的线性表C，并要求利用原表（即A表和B表）的结点空间构造C表。

【分析】对两个或两个以上结点按元素值递增/减排列的单链表进行操作时，应采用“指针平行移动、一次扫描完成”的策略。

【解答】算法如下：

```
void reverse_merge(LinkList &A, LinkList &B, LinkList &C)//把元素递增排列的链表A和B合并为C，
                                                         //且C中元素递减排列，使用原空间
{
    pa=A->next;pb=B->next;pre=NULL; //pa和pb分别指向A，B的当前元素
    while(pa || pb)
    {
        if(pa->data<pb->data || !pb)
        {
            pc=pa;q=pa->next;pa->next=pre;pa=q; //将A的元素插入新表
        }
```

```
        else
        {
            pc=pb;q=pb->next;pb->next=pre;pb=q; //将B的元素插入新表
        }
        pre=pc;
    }
    C=A;A->next=pc; //构造新表头
}//reverse_merge
```

22．假设以两个元素依值递增有序排列的线性表*A*和*B*分别表示两个集合（即同一表中的元素值各不相同），现要求另辟空间构成一个线性表*C*，其元素为*A*和*B*中元素的交集，且表*C*中的元素也依值递增有序排列。试对顺序表编写求*C*的算法。

【分析】类似21题，设置两个指针，再同时平行移动。

【解答】算法如下：

```
void SqList_Intersect(SqList A，SqList B，SqList &C)//求元素递增排列的线性表A和B的元素的交集并
                                                    //存入C中
{
    i=1;j=1;k=0;
    while(A.elem[i]&&B.elem[j])
    {
        if(A.elem[i]<B.elem[j]) i++;
        if(A.elem[i]>B.elem[j]) j++;
        if(A.elem[i]==B.elem[j])
        {
           C.elem[++k]=A.elem[i]; //当发现一个在A和B中都存在的元素，就添加到C中
           i++;j++;
        }
    }//while
}//SqList_Intersect
```

23．要求同22题，试对单链表编写求*C*的算法。

【分析】类似22题。注意对比顺序表和链表操作中的异同点，并从其不同的数据存储结构方面理解。

【解答】算法如下：

```
void LinkList_Intersect(LinkList A，LinkList B，LinkList &C)//在链表结构上重做上题
{
    p=A->next;q=B->next;
    pc=(LNode*)malloc(sizeof(LNode));
    while(p&&q)
    {
        if(p->data<q->data) p=p->next;
        else if(p->data>q->data) q=q->next;
            else
            {
                s=(LNode*)malloc(sizeof(LNode));
```

```
                s->data=p->data;
                pc->next=s;pc=s;
                p=p->next;q=q->next;
            }
    }//while
    C=pc;
}//LinkList_Intersect
```

24．对22题的条件作以下两点修改，对顺序表重新编写求得表*C*的算法。

（1）假设在同一表（*A*或*B*）中可能存在值相同的元素，但要求新生成的表*C*中的元素值各不相同；

（2）利用*A*表空间存放表*C*。

【分析】（1）可选择先求交集再删除相同元素，也可以先删除多余元素再求交集，或者在求交集的过程中直接跳过相同的元素。

（2）要求在*A*表空间存放表*C*，只要多一个辅助下标*k*即可。

【解答】算法如下：

```
void SqList_Intersect_True(SqList &A，SqList B)// 求元素递增排列的线性表A和B的元素的交集
                                                // 并存回A中
{
    i=1;j=1;k=0;
    while(A.elem[i]&&B.elem[j])
    {
        if(A.elem[i]<B.elem[j]) i++;
        else if(A.elem[i]>B.elem[j]) j++;
        else if(A.elem[i]!=A.elem[k])
        {
            A.elem[++k]=A.elem[i]; //当发现一个在A和B中都存在的元素
            i++;j++; //且C中没有，就添加到C中
        }
    }//while
    while(A.elem[k]) A.elem[k++]=0;
}//SqList_Intersect_True
```

25．对22题的条件作以下两点修改，对单链表重新编写求得表*C*的算法。

（1）假设在同一表（*A*或*B*）中可能存在值相同的元素，但要求新生成的表*C*中的元素值各不相同；

（2）利用原表（*A*表或*B*表）中的结点构造表*C*，并释放*A*表中的无用结点空间。

【分析】实现24、25题操作不应先分别删除表*A*和表*B*中多余的值相同的元素，应同样采用和21题相同的策略，只要进一步考虑，和表*B*中结点值相同的表*A*的结点值，是否和表*C*中当前最后一个结点的值相同即可。

【解答】算法如下：

```
void LinkList_Intersect_True(LinkList &A，LinkList B)//在链表结构上重做上题
{
```

```
    p=A->next;q=B->next;pc=A;
    while(p&&q)
    {
        if(p->data<q->data) p=p->next;
        else if(p->data>q->data) q=q->next;
        else if(p->data!=pc->data)
        {
           pc=pc->next;
           pc->data=p->data;
           p=p->next;q=q->next;
        }
     }//while
}//LinkList_Intersect_True
```

26. 已知*A*、*B*和*C*为3个递增有序的线性表，现要求对*A*表作如下操作：删去那些既在*B*表中出现又在*C*表中出现的元素。试对顺序表编写实现上述操作的算法，并分析你的算法的时间复杂度（注意：题中没有特别指明同一表中的元素值各不相同）。

【分析】先从*B*和*C*中找出共有元素，记为same，再在*A*中从当前位置开始，凡小于same的元素均保留（存到新的位置），等于same的就跳过，到大于same时就再找下一个same。

【解答】算法如下：

```
void SqList_Intersect_Delete(SqList &A, SqList B, SqList C)
{
    i=0;j=0;k=0;m=0; //i指示A中元素原来的位置，m为移动后的位置
    while(i<A.length&&j<B.length&& k<C.length)
    {
        if(B.elem[j]<C.elem[k]) j++;
        else if(B.elem[j]>C.elem[k]) k++;
        else
        {
            same=B.elem[j];   //找到相同元素same
            while(B.elem[j]==same) j++;
            while(C.elem[k]==same) k++; //j，k后移到新的元素
            while(i<A.length&&A.elem[i]<same)
                A.elem[m++]=A.elem[i++];  //需保留的元素移动到新位置
            while(i<A.length&&A.elem[i]==same) i++; //跳过相同的元素
        }
    }//while
    while(i<A.length)
        A.elem[m++]=A.elem[i++]; //A的剩余元素重新存储
  A.length=m;
}// SqList_Intersect_Delete
```

该算法的时间复杂度为$O(n)$。

27. 要求同26题，试对单链表编写算法，请释放*A*表中的无用结点空间。

【解答】算法如下：

```
void LinkList_Intersect_Delete(LinkList &A, LinkList B, LinkList C)//在链表结构上重做上
```

```
题
{
    p=B->next;q=C->next;r=A-next;
    while(p&&q&&r)
    {
        if(p->data<q->data) p=p->next;
        else if(p->data>q->data) q=q->next;
        else
        {
            u=p->data; //确定待删除元素u
            while(r->next->data<u) r=r->next; //确定最后一个小于u的元素指针r
            if(r->next->data==u)
            {
                s=r->next;
                while(s->data==u)
                {
                    t=s;s=s->next;free(t); //确定第一个大于u的元素指针s
                }//while
                r->next=s; //删除r和s之间的元素
          }//if
          while(p->data=u) p=p->next;
          while(q->data=u) q=q->next;
       }//else
    }//while
}//LinkList_Intersect_Delete
```

28．假设某个单向循环链表的长度大于1，且表中既无头结点也无头指针，已知s为指向链表中某个结点的指针，试编写算法在链表中删除指针s所指结点的前驱结点。

【分析】关键是找到s的前驱的前驱p，从而将p的Next指针指向s。

【解答】算法如下

```
Status Delete_Pre(CiLNode *s)//删除单循环链表中结点s的直接前驱
{
    p=s;
    while(p->next->next!=s) p=p->next; //找到s的前驱的前驱p
    p->next=s;
    return OK;
}//Delete_Pre
```

29．已知有一个单向循环链表，其每个结点中含3个域：prior，data和next，其中data为数据域，next为指向后继结点的指针域，prior也为指针域，但它的值为空（NULL），试编写算法将此单向循环链表改为双向循环链表，即使prior成为指向前驱结点的指针域。

【解答】算法如下：

```
Status DuLNode_Pre(DuLinkList &L)//完成双向循环链表结点的pre域
{
    for(p=L;!p->next->pre;p=p->next) p->next->pre=p;
    return OK;
}//DuLNode_Pre
```

30．已知由一个线性链表表示的线性表中含有3类字符的数据元素（如：字母字符、数

字字符和其他字符），试编写算法将该线性链表分割为3个循环链表，其中每个循环链表表示的线性表中均只含一类字符。

【分析】先设置3个空的循环链表，然后将单链表中的结点分别插入这3个链表。注意3个结果表的头指针在参数表中应设置为变参。

【解答】算法如下：

```
Status LinkList_Divide(LinkList &L, CiList &A, CiList &B, CiList &C)
        //把单链表L的元素按类型分为3个循环链表。CiList为带头结点的单循环链表类型
{
    s=L->next;
    A=(CiList*)malloc(sizeof(CiLNode));p=A;
    B=(CiList*)malloc(sizeof(CiLNode));q=B;
    C=(CiList*)malloc(sizeof(CiLNode));r=C; //建立头结点
    while(s)
    {
        if(isalphabet(s->data))
        {
           p->next=s;p=s;
        }
        else if(isdigit(s->data))
        {
            q->next=s;q=s;
        }
        else
        {
            r->next=s;r=s;
        }
    }//while
    p->next=A;q->next=B;r->next=C; //完成循环链表
}//LinkList_Divide
```

在31~33题中，“异或指针双向链表”类型XorLinkedList和指针异或函数XorP定义为：

```
typedef struct XorNode {
    char data;
    struct XorNode LRPtr;
} XorNode,  *XorPointer;
typedef struct {            //无头结点的异或指针双向链表
    XorPointer Left,  Right;     //分别指向链表的左端和右端
}XorLinkedList;
XorPointer  XorP (XorPointer p,  XorPointer q ) ;
//指针异或函数XorP返回指针p和q的异或（XOR）值
```

31．假设在算法描述语言中引入指针的二元运算“异或”（用“⊕”表示），若a和b为指针，则a⊕b的运算结果仍为原指针类型，且

a⊕(a⊕b)=(a⊕a)⊕b=b

(a⊕b)⊕b=a⊕(b⊕b)=a

则可利用一个指针域来实现双向链表*L*。链表*L*中的每个结点只含两个域：data域和LRPtr域，其中，LRPtr域存放该结点的左邻与右邻结点指针（不存在时为NULL）的异或。若设指针L.Left指向链表中的最左结点，L.Right指向链表中的最右结点，则可实现从左向右或从右向左遍历此双向链表的操作。试写一算法按任一方向依次输出链表中各元素的值。

【分析】设指针p指向当前结点，left指向它的左邻结点，right指向它的右邻结点，则有right == left ⊕ p->LRPtr和left == p->LRPtr ⊕ right。

一般而言，设指针r的初值为NULL。若从左到右遍历，则p的初值为链表的左端指针L.Left；若从右到左遍历，p的初值为链表的右端指针L.Right。访问p结点后，下一个结点的指针q = r ⊕ LRPtr或q = p ->LRPtr ⊕ r。

【解答】算法如下：

```
void Print_XorLinkedList(XorLinkedList L)//从左向右输出异或链表的元素值
{
    p=L.left;pre=NULL;
    while(p)
    {
        printf("%d", p->data);
        q=XorP(p->LRPtr, pre);
        pre=p;p=q; //任何一个结点的LRPtr域值与其左结点指针进行异或运算即得到其右结点指针
    }
}//Print_XorLinkedList
```

32. 采用31题所述的存储结构，写出在第*i*个结点之前插入一个结点的算法。

【分析】32与33题类似，均是在深入理解“异或指针双向链表”的数据结构特点的基础上，对链表基本操作的一种类比。

【解答】算法如下：

```
Status Insert_XorLinkedList(XorLinkedList &L, int x, int i)//在异或链表L的第i个元素前插入元素x
{
    p =L.left;pre= NULL;
    r =(XorNode*)malloc(sizeof(XorNode));
    r->data=x;
    if(i==1) //插入点在最左边的情况
    {
        p->LRPtr =XorP(p.LRPtr, r);
        r->LRPtr =p;
        L.left =r;
        return OK;
    }
    j=1;q=p->LRPtr; //插入点在中间的情况
    while(++j<i&&q)
    {
        q=XorP(p->LRPtr, pre);
        pre=p;p=q;
    }//while //在p, q两结点之间插入
```

```
    if(!q) return INFEASIBLE; //i不可以超过表长
    p->LRPtr=XorP(XorP(p->LRPtr, q), r);
    q->LRPtr=XorP(XorP(q->LRPtr, p), r);
    r->LRPtr=XorP(p, q); //修改指针
    return OK;
}//Insert_XorLinkedList
```

33．采用31题所述的存储结构，写出删除第i个结点的算法。

【解答】算法如下：

```
Status Delete_XorLinkedList(XorLinkedList &L, int i)//删除异或链表L的第i个元素
{
    p=L.left;pre=NULL;
    if(i==1) //删除最左结点的情况
    {
        q=p->LRPtr;
        q->LRPtr=XorP(q->LRPtr, p);
        L.left=q;free(p);
        return OK;
    }
    j=1;q=p->LRPtr;
    while(++j<i&&q)
    {
        q=XorP(p->LRPtr, pre);
        pre=p;p=q;
    }//while //找到待删结点q
    if(!q) return INFEASIBLE; //i不可以超过表长
    if(L.right==q) //q为最右结点的情况
    {
        p->LRPtr=XorP(p->LRPtr, q);
        L.right=p;free(q);
        return OK;
    }
    r=XorP(q->LRPtr, p); //q为中间结点的情况，此时p、r分别为其左、右结点
    p->LRPtr=XorP(XorP(p->LRPtr, q), r);
    r->LRPtr=XorP(XorP(r->LRPtr, q), p); //修改指针
    free(q);
    return OK;
}//Delete_XorLinkedList
```

34．设以带头结点的双向循环链表表示的线性表$L=(a_1, a_2, \cdots, a_n)$。试写一时间复杂度为$O(n)$的算法，将$L$改造为$L=(a_1, a_3, \cdots, a_n, \cdots, a_4, a_2)$。

【分析】next链和pre链的调整只能分开进行。如同时进行调整的话，必须使用堆栈保存偶数结点的指针，否则将会破坏链表结构，造成结点丢失。

【解答】算法如下：

```
void OEReform(DuLinkedList &L)//按1, 3, 5, …, 4, 2的顺序重排双向循环链表L中的所有结点
{
    p=L.next;
```

```
    while(p->next!=L&&p->next->next!=L)
    {
         p->next=p->next->next;
         p=p->next;
    } //此时p指向最后一个奇数结点
    if(p->next==L) p->next=L->pre->pre;
    else p->next=l->pre;
    p=p->next; //此时p指向最后一个偶数结点
    while(p->pre->pre!=L)
    {
        p->next=p->pre->pre;
        p=p->next;
    }
    p->next=L; //按题目要求调整了next链的结构，此时pre链仍为原状
    for(p=L;p->next!=L;p=p->next) p->next->pre=p;
    L->pre=p; //调整pre链的结构，同32题方法
}//OEReform
```

35. 设有一个双向循环链表，每个结点中除有prior、data和next 3个域外，还增设了一个访问频度域freq。在链表被起用之前，频度域freq的值均初始化为零，而每当对链表进行一次Locate(L，x)的操作后，被访问的结点（即元素值等于x的结点）中的频度域freq的值便增加1，同时调整链表中结点之间的次序，使其按访问频度非递增的次序顺序排列，以便始终保持被频繁访问的结点总是靠近表头结点。试编写符合上述要求的Locate操作的算法。

【分析】关键的操作一是对频域的累加，再就是每次查找后进行按频域的排序，其余的便和链表的Locate操作相似。

【解答】算法如下：

```
DuLNode * Locate_DuList(DuLinkedList &L, int x)//带freq域的双向循环链表上的查找
{
    p=L.next;
    while(p.data!=x&&p!=L) p=p->next;
    if(p==L) return NULL; //没找到
    p->freq++;q=p->pre;
    while(q->freq<=p->freq) q=q->pre; //查找插入位置
    if(q!=p->pre)
    {
        p->pre->next=p->next;p->next->pre=p->pre;
        q->next->pre=p;p->next=q->next;
        q->next=p;p->pre=q; //调整位置
    }
    return p;
}//Locate_DuList
```

在36~37题中，稀疏多项式采用的顺序存储结构SqPoly定义为：

```
typedef struct {
    int coef;
```

```
    int exp
} PolyTerm;
typedef struct {     //多项式的顺序存储结构
    PolyTerm   *data;
    int         length;
} SqPoly;
```

36．已知稀疏多项式$P_n(X)=c_1x^{e_1}+c_2x^{e_2}+\Lambda+c_mx^{e_m}$，其中$n=e_m>e_{m-1}>\cdots>e_1\geqslant 0$，$c_i\neq 0$（$i=1,2,\cdots,m$），$m\geqslant 1$。试采用存储量同多项式项数$m$成正比的顺序存储结构，编写求$P_n(x_0)$的算法（$x_0$为给定值），并分析你的算法的时间复杂度。

【分析】只存储c_i和e_i（$i=1,2,\cdots,m$），则无论顺序结构或链表结构，都符合题目的要求。注意算法时间复杂度应是$O(e_m)$，而不是$O(\sum_{i=1}^{m}e_i)$。

【解答】算法如下：

```
float GetValue_SqPoly(SqPoly P, int x0)//求升幂顺序存储的稀疏多项式的值
{
    PolyTerm*q;
    xp=1;q=P.data;
    sum=0;ex=0;
    while(q->coef)
    {
        while(ex<q->exp) xp*=x0;
        sum+=q->coef*xp;
        q++;
    }
    return sum;
}//GetValue_SqPoly
```

37．采用36题给定的条件和存储结构，试编写求$P(x)=P_{n_1}(x)-P_{n_2}(x)$的算法，将结果多项式存放在新辟的空间中。

【解答】算法如下：

```
void Subtract_SqPoly(SqPoly P1, SqPoly P2, SqPoly &P3)//求稀疏多项式P1减P2的差式P3
{
    PolyTerm *p, *q, *r;
    Create_SqPoly(P3); //建立空多项式P3
    p=P1.data;q=P2.data;r=P3.data;
    while(p->coef&&q->coef)
    {
        if(p->exp<q->exp)
        {
            r->coef=p->coef;
            r->exp=p->exp;
            p++;r++;
        }
        else if(p->exp<q->exp)
```

```
            {
                r->coef=q->coef;
                r->exp=q->exp;
                q++;r++;
            }
            else
            {
                if((p->coef-q->coef)!=0) //只有同次项相减不为零时才需要存入P3中
                {
                    r->coef=p->coef-q->coef;
                    r->exp=p->exp;r++;
                }//if
                p++;q++;
            }//else
        }//while
        while(p->coef) //处理P1或P2的剩余项
        {
            r->coef=p->coef;
            r->exp=p->exp;
            p++;r++;
        }
        while(q->coef)
        {
            r->coef=q->coef;
            r->exp=q->exp;
            q++;r++;
        }
    }//Subtract_SqPoly
```

在38~39题中，稀疏多项式采用的循环链表存储结构LinkedPoly定义为：

```
typedef  struct  PolyNode {
    PolyTerm  data;
    Struct PolyNode  *next;
} PolyNode,  *PolyLink;
typedef  PolyLink  LinkedPoly;
```

38．试以循环链表作稀疏多项式的存储结构，编写求其导函数的算法，要求利用原多项式中的结点空间存放其导函数（多项式），同时释放所有无用（被删）结点。

【分析】考查对线性表基本操作的组合，注意对于常数项求导时要删除结点。

【解答】算法如下：

```
void QiuDao_LinkedPoly(LinkedPoly &L)//对有头结点循环链表结构存储的稀疏多项式L求导
{
    p=L->next;
    if(!p->data.exp)
    {
        free(L->next);p=p->next;L->next=p;
    }
    while(p!=L)
```

```
    {
        p->data.coef*=p->data.exp--;//对每一项求导
        p=p->next;
    }
}//QiuDao_LinkedPoly
```

39. 试编写算法，将一个用循环链表表示的稀疏多项式分解成两个多项式，使这两个多项式中各自仅含奇次项或偶次项，并要求利用原链表中的结点空间构成这两个链表。

【分析】与30题类似。由于此题只拆成两个表，因此也可只设一个奇次项链表的头结点，并构成一个空的循环链表，然后将所有的奇次项结点从原表中删去并插入至新的链表中。

【解答】算法如下：

```
void Separation (LinkedPoly &plyn, LinkedPoly &odd )
{
// plyn为指向表示稀疏多项式的循环链表头结点的指针，
//odd为新产生的仅含奇次项链表的头指针，运算后plyn链表中仅含偶次项
    odd= (LinkedPoly ) malloc ( sizeof (Poly Node ) );
    odd->next=odd;      //建立奇次项空表
    q=plyn;  p=plyn->next;  s=odd;    //s指向奇次项链表的尾结点
    while (p! = plyn)
    {
        if (p->exp % 2 = = 0) {q=p;  p=p->next; }
        else
        {
            q->next=p->next;          //从原表中删去奇次项结点
            p->next=s->next; s->next=p;     //插入至新表中
            p=q->next; s=s->next;
        } //else
    } //while
} //split
```

2.4　考研真题分析

例题2-1　（清华大学1998年试题）

线性表是具有n个（　　）的有限序列。

A. 表元素　　B. 字符　　C. 数据元素　　D. 数据项　　E. 信息项

【分析】　注意区分不同的概念，数据元素是数据的基本单位；数据元素可由若干数据项组成。

【解答】C。

例题2-2　（中国科学技术大学1998年试题）

将两个各有n个元素的有序表归并成一个有序表，其最少的比较次数是_____。

A. n　　B. $2n-1$　　C. $2n$　　D. $n-1$

【分析】　当一个表的最小元素大于另一个表的最大元素时，比较次数为n次。

【解答】A。

例题2-3　（北京航空航天大学1998年试题）

在非空双向循环表中q所指的结点后面插入p所指的结点的过程已经依次进行了3步：p.llink:=q;p.rlink:=q.rlink;q.rlink:=p；第4步应是什么动作？（写出该动作对应的语句。）

【解答】　q.rlink.llink:=p;

例题2-4　（北京航空航天大学1998年试题）

若较频繁地对一个线性表进行插入和删除操作，该线性表宜采用何种存储结构？为什么？

【解答】　宜采用链式存储结构。因为采用链式存储结构的线性表，插入和删除操作时间复杂度为$O(1)$，而采用顺序存储结构的线性表插入和删除操作时间复杂度为$O(n)$。

例题2-5　（北京邮电大学1998年试题）

判断正误：

① 数据元素是数据的基本单位；数据元素可以由数据项组成；数据项是数据的最小单位。

② 结点内部的存储空间应该是连续的。

【解答】　① 错误　② 错误

例题2-6　（北京邮电大学1998年试题）

表长为n的顺序存储的线性表，当在任何位置上插入或删除一个元素的概率相等时，插入一个元素所需移动元素的平均个数为__（1）__，删除一个元素需要移动元素的平均个数为__（2）__。

A. $(n-1)/2$　　B. n　　C. $n+1$　　D. $n-1$

E. $n/2$　　F. $(n+1)/2$　　G. $(n-2)/2$

【分析】

插入一个元素所需移动元素的平均个数

$$E_i=\sum_{i=1}^{n+1}\frac{1}{n+1}(n-i+1)=\frac{n}{2}$$

删除一个元素所需移动元素的平均个数

$$E_d=\sum_{i=1}^{n}\frac{1}{n}(n-i)=\frac{n-1}{2}$$

【解答】（1）E　　（2）A

例题2-7　（北京邮电大学1998年试题）

算法改错：请将下面算法中的错误及其语句标号写在答卷上，并写上改正后的语句。

算法说明：现有两个带表头结点的单链表l_a和l_b，此两个表本身无相同元素存在，l_a表递增有序，l_b表递减有序，且l_b表中的元素包含在l_a表连续的某一部分中，如图2-1所示。算法要找到这部分元素在l_a表中的位置，并将它们逆置，使之与l_b表中的顺序相同。

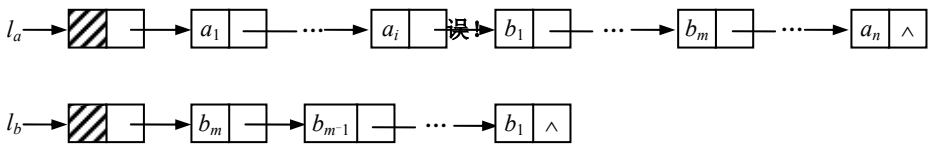


图 2-1　单链表 l_a 和 l_b

```
TYPE
     pointer=^node;
     node=RECORD
          data:datatype;
          next:pointer
     END;
     stack=ARRAY[1..m]OF pointer;
  PROCEDURE matchinvert(VAR la:pointer;lb:pointer);
     VAR top:0..m;
         p, q, u, s, t:pointer;
(1)     BEGIN
(2)         q:=lb^.next;
(3)         u:=la;
(4)         p:=la^.next;
(5)         top:=0;
(6)         IF(q=NIL)OR(p=NIL)THEN return;
(7)         WHILE q^.next<>NIL DO
(8)         BEGIN
(9)             top:=top+1;
(10)            s[top]:=1;
(11)            q:=q^.next;
(12)        END;
(13)        WHILE(top<>0)AND(p<>NIL)DO
(14)           IF p^.data=s[top]THEN
(15)           BEGIN
(16)              p:=p^.next;
(17)              top:=top+1;
(18)           END
(19)           ELSE BEGIN
```

```
(20)            u:=u^.next;
(21)            p:=u^.next;
(22)          END;
(23)        IF top<>0 THEN
(24)          write('not match');
(25)        ELSE BEGIN
(26)           s:=u^.next;
(27)           u^.next:=NIL;
(28)          WHILE u<>p DO
(29)          BEGIN
(30)             t:=s;
(31)             s:=s^.next;
(32)             t^.next:=u^.next;
(33)             u^.next:=s;
(34)           END
(35)        END
(36)    END
```

【分析】算法分为3个部分：

（7）~（12）将表l_b的内容读到数组stack中去；

（13）~（22）用stack中的内容去匹配l_a；

（23）~（35）将匹配后的表逆置。

【解答】共有7处需要修改：

（7）　WHILE q<>NIL DO

（10）　stack[top]:=q;

（14）　IF p^.data=stack[top]^.data THEN

（17）　top:=top-1;

（27）　u^.next:=p;

（28）　WHILE s<>p DO

（33）　u^.next=t;

例题2-8　（北京工业大学1998年试题）

写出在双向链表da中的插入操作算法，算法中插入位置的获取可直接引入getnodep(da，i)，其中参数da为双向链表，i是要插入的数据，要求算法中含有双向链表da的结点结构描述。

【解答】　算法如下：

```
TYPE link=↑node;
       node=RECORD
          data:element;
          pre，next:link;
       END;
PROCEDURE insert(VAR da:link;x:elemtype);
VAR p，q:link;
BEGIN
     p:=getnodep(da，x);
```

```
    new(q);
    q↑.data:=x;
    q↑.next:=p↑.next;
    p↑.next:=q;
    q↑.next↑.pre:=q;
    q↑.pre:=p;
END.
```

例题2-9　（中国科学院计算机网络信息中心2000年试题）

已知有如下定义的静态链表：

```
TYPE component=Record
                data:elemtp;
                next:0..maxsize;
                End
var stalist:array[0..maxsize]of component;
```

以及3个指针：av指向头结点，p指向当前结点，pre指向p的前驱结点。

现要求修改静态链表中next域中的内容，使得该静态链表具有双向链表的功能，从当前结点p既能往后查找，也能往前查找：

（1）定义next域中的内容（用老的next域中的值表示）。

（2）如何得到当前结点p的前驱（pre）的前驱，给出计算式。

（3）如何得到p的后继，给出计算式。

【解答】

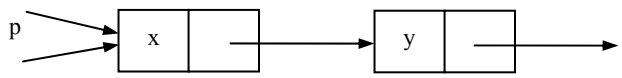
（1）stalist[p].next=stalist[p].next-pre

（2）p-stalist[pre].next

（3）stalist[p].next+pre

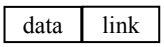
例题2-10　（华北计算技术研究所2001年试题）

已知如下单链表：



x、y均为整数，要求不使用中间变量，写出实现x、y互换的简要操作。

结点结构为：



【解答】　基本思想：以指针为基础，以x和y的算术运算为变化条件。

```
p↑.data=p↑.data+p↑.link↑.data
p↑.link↑.data=p↑.data-p↑.link↑.data
p↑.data=p↑.data-p↑.link↑.data
```

例题2-11（中国人民大学2002年试题）

设有一单向循环链表L1，对其遍历的结果是：x_1，x_2，x_3，…，x_{n-1}，x_n。请你将该循环

链表拆成两个单向循环链表L1和L2。使得：

当n为偶数时，对L1的遍历结果为：x_1，x_3，…，x_{n-1}；

对L2的遍历结果为：x_n，x_{n-2}，…，x_4，x_2；

当n为奇数时，对L1的遍历结果为：x_1，x_3，…，x_n；

对L2的遍历结果为：x_{n-1}，x_{n-3}，…，x_4，x_2。

【解答】算法如下：

```
struct node{datatype  data;
          struct node*  next;
          };
struct node* depatch {struct node*L1}
{  struct node *L2, *p1, *p2a, *p2b, *q;
   int i;
   L2=L1->next;  L1->next=L1;  q=L2->next;
   L2->next=L2;  p1=L1;   p2a=p2b=L2;
   i=3;
   while (q!=L1)
   { if(i%2 = =1)
     { p1->next=q; p1=q; q=q->next;  i++;  p1->next=L1;}
     else
     { L2=q; q=q->next; i++; L2->next=p2b;  p2b=L2;  p2a->next=L2;}
   }
}
```

例题2-12 （北京科技大学2002年试题）

设单链表中结点的数据域为data，指针域为next，指针p为表中某一结点的地址，请写出在p结点之前插入一个s结点的C语言描述语句。

【解答】 s↑next:=p;

例题2-13 （北京科技大学2002年试题）

设L为单向循环链表（不带头结点）第一结点的指针，结点编号分别为1，2，…，n，从链表中编写为k($1\leqslant k\leqslant n$)的结点开始计有选举权，计到m($1\leqslant m\leqslant n$)时的结点出列(删除)，再从列的下一结点1开始计数，计到m时的结点又出列……依此类推，直到表中所有的结点都出列为止。请用C语言函数形式写出完成任务的算法Tosephu(L，n，k，m)。

【解答】 算法如下：

```
Tosephu(struct node *L, int n, int k, int m)
{    int i;
     struct node *p, *pre;
     pre=L;p=pre->next;
     i=2;
     while(i<k)
     {    pre=p;p=p->next;i++;}
     while(p->next!=p)
          {i=1;
```

```
                while(i<m)
                {pre=p;p=p->next;i++;}
            printf("%d", p->data);
            pre->next=p->next;
            free(p);
            p=pre->next;
        }
        printf("%4d", p->data);
        free(p);
    }
```

例题2-14 （北京科技大学2002年试题）

设有n个顶点的有向图G已用邻接表结构存储，顶点表指针为g，且图中各顶点的入度已记录在顶点的id域中（即g->data[i].id=第i($1\leqslant i\leqslant n$)个顶点的入度）。请用C语言函数形式写出判断图G是否存在回路的算法Top_cycle(g, n)（注：此算法中可调用栈操作的基本算法）。

【解答】　算法如下：

```
int Top_cycle(Graph g, int n)
{    int i, j, m; struct rexnode *p;
     int visited[MAXN];
     for(i=0;i<n;i++)
     {if(g.data[i], id>0)
          {for(j=0;j<n;j++)
               visited[j]=0;
          clear stack(s);
          push(s, i);
          while(!Empty stack(s))
          {
               m=pop(s);
               if(m==i && visited[i])return 1;
               visited[i]=1;
               p=g.data[i].firsarc;
               while(p!=NULL)
               {     if(!visited[p->num])
                    {    visited[p->num]=1;
                         push(s, p->num);
                         p=p->next arc;
                    }//if
               }//while
          }//while
      }//if
  }//for
  return 0;
}
```

例题2-15 （北京科技大学2002年试题）

算法设计题：

设有两个带头结点的单链表A和B，链表中结点的数据域为data（设为整型），指针域为next。请用C语言函数形式写出将表A和B合并为一个单链表L的算法Union(A，B，L)（注：若表A和B中有数据值相同的结点，只保留其中一个）。

【解答】 算法如下：

```
void Union (struct node *A, struct node *B, struct node *L)
{    struct node *pa, *pb, pl;
     int flag;
     L->next=NULL;pa=A->next;pb=B->next;pl=L->next;
     while(pa!=NULL)
     {    flag=0
          while(pl!=NULL)
          {    if (pl->data==pa->data){flag=1;break;}
               pl=pl->next;
          }
          if(!flag)
          {    A->next=pa->next;pa->next=L->next;L->next=pa;pa=A->next;}
          else
          {    free(pa);pa=A->next;}
     }
     while(pb!=NULL)
     {  将上一个while循环中的pa全换成pb，A全换成B  }
}
```

例题2-16 （武汉理工大学2002年试题）

选择题：

指针P所指的元素是双向循环链表L的尾元素的条件是___A___；若队列采用链式存储，则该链式队列___B___。

供选择的答案：

A：①P=L　②P=NULL　③P↑.Link=L　④P↑.Rlink=L

B：①存在队满的情况　②不存在队空的情况

③入队之前必须判断队满否　④出队之前必须判断队空否

【解答】 A: ④　B:④

例题2-17 （武汉理工大学2002年试题）

判断题：

一个循环链表可以由所给定的头指针或者尾指针惟一地确定。

【解答】 √

例题2-18 （武汉理工大学2002年试题）

算法设计题：

写一算法，建立双向循环链表。

【解答】设双向循环链表的结点为

```
struct DNode {
     ElemType Data;
     struct DNode *left;
     struct DNode *right;
}
```

设向双向循环链表的头部插入一个结点的函数为

```
void Insert DUL(DNode *&HL, ElemType x)
{
     DNode *p=(DNode *) malloc(sizeof (DNode));
     p->data:=x;
     p->Pright:=HL;
     p->left:=HLleft;
     if(HL->left !=NULL)
          HL->left->right:=p;
     HL:=p;
}
```

而构建一个双向循环链表的方法为

```
struct DNode *CreateDul(ElemType A[ ], int n);
{    struct DNode *HL;
     HL:=NULL;HL↑.right:=HL↑.left:=Null;
     for(i=0;i<n;i++) InsertDul(HL, A[i]);
     return HL;
}
```

例题2-19 （南京理工大学2002年试题）

简答题：

写出在双向链表指针p之后插入结点s的操作序列。

【解答】

（1）s->right=p->right （2）if(p->right)p->right->left=s （3）s->left=p （4）p->right=s

例题2-20 （南京理工大学2002年试题）

选择题：

在一个单链表中，若删除*p*结点的后继结点，则_____。

A. p->next=p->next->next;

B. p=p->next;p->next=p->next->next;

C. p->next=p->next;

D. p=p->next->next;

【解答】A

2.5 自 测 题

自测题2-1

设A是一个线性表（a_1，a_2，…，a_n），若采用顺序存储结构，则在等概率的前提下，平均每插入一个元素需要移动的元素个数为多少？若元素插在a_i与a_{i+1}之间（$0 \leqslant i \leqslant n-1$）的概率为$\frac{n-i}{n(n+1)/2}$，则平均每插入一个元素所要移动的元素个数又是多少？

自测题2-2

已知3个带头结点的线性链表A、B和C中的结点均依元素值自小至大非递减排序（可能存在两个以上值相同的结点），编写算法对A表进行如下操作：使操作后的链表A中仅留下3个表中均包含的数据元素的结点，且没有值相同的结点，并释放所有无用结点。限定算法的时间复杂度为$O(m+n+p)$，其中m、n和p分别为3个表的长度。

自测题2-3

已知l_a是带头结点的单链表的头指针，试编写逆序输出表中各元素的递归算法。

自测题2-4

请在下列算法的方框内填入适当的语句：

```
FUNCTION inclusion(ha, hb:linklisttp):boolean;
{以ha和hb为头指针的单链表分别表示有序表A和B，本算法判别表A是否包含在表B内；若是，则返回
“true”，否则返回“false”}
BEGIN
    pa:=ha^.next;
    pb:=hb^.next;
        (1)        ;
    WHILE    (2)    DO
    IF pa^.data=pb^.data
    THEN     (3)
    ELSE     (4)
        (5)
END;
```

自测题2-5

已知结点指针p、q分别表示双链表中任意两个相邻结点（即p^.rlink=q且q^.link=p），请写出删除q所指结点的Pascal程序段。

自测题2-6

已知顺序表中有m个记录，表中记录不依关键字有序排列。试编写算法为该顺序表建立一个有序的索引表，索引表中的每一项应含记录的关键字和该记录在顺序表中的序号，要求该算法的时间复杂度在最好的情况下能达到$O(m)$。

自测题2-7

假设有两个按元素值递增有序排列的线性表*A*和*B*，均以带头结点的单链表作为存储结构，试编写算法将*A*表和*B*表归并成一个按元素值递减有序排列的线性表*C*，并要求利用原表（*A*表和*B*表）的结点空间存放表*C*。

自测题2-8

写一算法，将一单链表逆置。要求逆置在原链表上进行，不允许重新构造一个链表。

自测题2-9

下面的程序段是合并两条链（*f*和*g*）为一条链*f*的过程。作为参数的两条链都是按结点number值由大到小链接的，合并后新链仍按此方式链接。请填写下述空框，使程序能正确运行。

```
TYPE pointer=^node;
     node=RECORD
       number:integer;
       next:pointer;
     END;
PROCEDURE  combine(VAR  f:pointer;g:pointer);
VAR  h,  p:pointer;
BEGIN
     new(h);
     h^.next:=NIL;
     p:=h;
     WHILE(f<>NIL)AND(g<>NIL)DO
       IF  f^.number >=g^.number THEN
       BEGIN
           p^.next:=  A  ; p:=  B  ;  C  ;
       END
       ELSE BEGIN
          p^.next:=  D  ; p:=  E  ;  F
       END;
     IF f=NIL  THEN  G  ;
     IF g=NIL  THEN  H  ;
     f:=h^.next;
     dispose(h);
END
```

2.6　自测题答案

【自测题2-1】

在等概率的前题下，平均每插入一个元素需要移动的元素个数为：

$$(0+1+2+\Lambda\ +n)/(n+1)=\frac{n}{2}$$

若元素插在a_i与a_{i+1}之间（$0 \leqslant i \leqslant n-1$）的概率为$\frac{n-i}{n(n+1)/2}$，则平均每插入一个元素所要移动的元素个数是$\sum_{i=0}^{n-1}\frac{(n-i)^2}{n(n+1)/2}=(2n+1)/3$。

【自测题2-2】

```
TYPE
     pointer=^node;
     node=RECORD
          data:datatype;
          next:pointer;
          END;
     linklist=pointer;

PROCEDURE MoveNext(VAR pre, p:linklist);
BEGIN
     pre^.next=p^.next
     dispose(p);
     p:=pre^.next;
     WHILE pre^.data=p^.data DO
     BEGIN
          pre^.next:=p^.next;
          dispose(p);
          p:=pre^.next;
     END;
END

PROCEDURE Skip(VAR pre, p:linklist);
BEGIN
     pre:=p;
     p:=p^.next;
     WHILE pre^.data=p^.data DO
     BEGIN
          pre^.next:=p^.next;
          dispose(p);
          p:=pre^.next;
     END;
END

PROCEDURE Conjunct(VAR HA, HB, HC:linklist);
VAR  PreA, PreB, PreC, PA, PB, PC, Pt:linklist;
BEGIN
     PreA:=HA;
     PreB:=HB;
     PreC:=HC;
     PA:=HA^.next;
     PB:=HB^.next;
     PC:=HC^.next;
```

```
WHILE PA<>NIL AND PB<>NIL AND PC<>NIL DO
IF PA^.data=PB^.data AND PB^.data=PC^.data THEN
BEGIN
     Skip(PreA, PA);
     PreB:=PB;
     PreC:=PC;
     PB:=PB^.next;
     PC:=PC^.next;
END
ELSE
     IF PA^.data<=PB^.data AND PA^.data<=PC^.data THEN
          MoveNext(PreA, PA);
     ELSE
          IF PB^.data AND PB^.data<=PC^.data THEN
          BEGIN
               PreB:=PB;
               PB:=PB^.next;
          END
          ELSE
               IF PC^.data <=PA^.data AND PC^.data<=PB^.data THEN
               BEGIN
                    PreC:=PC;
                    PC:=PC^.next;
               END;
     IF PA<>NIL THEN PreA^.next=NIL;
     WHILE PA<>NIL DO
     BEGIN
          PreA:=PA;
          PA:=PA^.next;
          Dispose(PreA);
     END
END
```

【自测题2-3】

```
PROCEDURE reverse(la:linklist);
BEGIN
     reverse0(la);
END;

PROCEDURE reverse0(l:linklist);
BEGIN
     IF  1<>NIL THEN
     BEGIN
          reverse0(1^.next);
          write(l^.data);
     END
END.
```

【自测题2-4】

可以考虑用递归来实现。

（1）IF pa=NIL THEN return(true)
（2）pb<>NIL AND pa^.data>=pb^.data
（3）return(inclusion(pa^.next, pb^.next));
（4）pb:=pb^.next;
（5）return(false);

【自测题2-5】

```
p^.rlink:=q^.rlink;
IF q^.rlink<>NIL THEN
  q->rlink->llink=p;
dispose(q);
```

【自测题2-6】

采用插入法建立索引。

```
typedef STRUCT INDEXELEM{
     KeyType   key;          //关键字
     int       sn;           //序号
}IndexType;                  //索引项

IndexType idx[1..m];  //索引表
DataType data[1..m];  //原顺序表
for(i=1;i<=m;i++){
     j=i-1;
     while(j>0 && idx[j].key>data[i].key){
          idx[j+1]=idx[j];
          j--;
     }
     idx[j+1].key=data[i].key;
     idx[j+1].sn=I;
}
```

【自测题2-7】

```
TYPE
     pointer=^node;
     node=RECORD
       data:datatype;
       next:pointer;
     END;
linklist=pointer;

PROCEDURE connect_down(VAR la,  lb,  lc:linklist);
{lc为指向一头结点的指针}
BEGIN
     pa:=la^.next;
     pb:=lb^.next;
     pc:=la;
     lc^.next:=NIL;
     dispose(lb);
```

```
    WHILE(pa<>NIL)AND(pb<>NIL)DO
         IF pa^.data<pb^.data THEN
              q:=pa^.next;
              pa^.next:=lc^.next;
              lc^.next:=pa;
              pa:=q;
         ELSE
              q:=pb^.next;
              pb^.next:=lc^.next;
              lc^.next:=pb;
              pb:=q;
         ENDIF
    ENDWHILE;

    IF pb=NIL THEN
         pb:=pa;
    ENDIF

    WHILE pb<>NIL DO
         q:=pb^.next;
         pb^.next:=lc^.next;
         lc^.next:=pb;
         pb:=q;
    ENDWHILE
END
```

【自测题2-8】

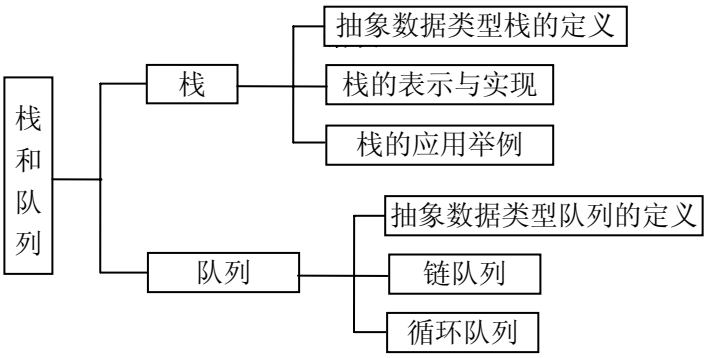
```
FUNCTION reverse(h)
BEGIN
     p:=h;
     q:=p^.next;
     p^.next:=NIL;
     IF(q^Next=NIL)THEN
     BEGIN
          q^.next:=p;
          return(q);
     END
     r:=q^.next;
     WHILE(r^.next!=NIL)DO
     BEGIN
          p^.next:=p;
          p:=q;
          q:=r;
          r:=r^.next;
     END
     r^.next:=q;
     return(r);
END
```

【自测题2-9】

A. f　　B. p^.next
C. f:=f^.next　　D. g
E. p^.next　　F. g:=g^.next
G. p^.next:=g　　H. p^next:=f

第3章　栈 和 队 列

3.1　基本知识结构图



3.2　知 识 点

1. 抽象数据类型栈的定义

栈是限定仅在表尾进行插入或删除操作的线性表，又称后进先出的线性表。

2. 栈的表示与实现

栈的存储通常采用顺序存储结构，即利用一组地址连续的存储单元依次存放自栈底到栈顶的数据元素，同时附设指针top指示栈顶元素在顺序栈中的位置。可以访问线性表中的数据元素，也可以对数据元素进行查找、插入、删除等操作，或者找出某数据元素的前驱、后继元素等。

3. 栈与递归的实现

一个直接调用自己或通过一系列的调用语句间接地调用自己的函数称为递归函数。

在递归函数的递归调用过程中，当有多个函数构成嵌套调用时，按照“后调用先返回”的原则，函数之间的信息传递和控制转移必须通过“栈”来实现。

4. 队列的定义

队列是一种先进先出的线性表，它只允许在表的一端进行插入，而在另一端删除元素。

5. 链队列

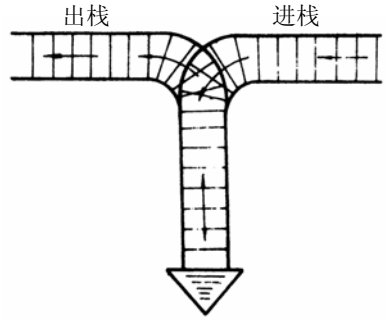
用链表表示的队列简称为链队列。一个链队列需要两个分别指示队头和队尾的指针惟一确定。

6. 循环队列

将顺序队列臆造为一个环状的空间，指针和队列元素之间关系不变，称之为循环队列。

3.3 习题及参考答案

1．若按下图所示铁道进行车厢调度（注意：两侧铁道均为单向行驶道），则请回答：



（1）如果进站的车厢序列为123，则可能得到的出站车厢序列是什么？

（2）如果进站的车厢序列为123456，则能否得到435612和135426的出站序列，并请说明为什么不能得到或者如何能得到（即写出以‘S’表示进栈和以‘X’表示出栈的栈操作序列）。

【解答】（1）123，132，213，231，321。

（2）可以得到135426，不可能得到435612，因为‘4356’出栈说明12已在栈中，则1不可能在2之前出栈。

2．简述栈和线性表的差别。

【解答】栈是限定仅在表尾进行插入或删除操作的线性表。

3．写出下列程序段的输出结果（栈的元素类型SElemType 为char）。

```
void main( ) {
     Stack S;
     char x, y;
     InitStack(S);
     x='c';  y='k';
     Push (S, x) ;  Push (S, 'a' );  Push (S, y) ;
     Pop (S, x) ;  Push (S, 't' ) ;  Push (S, x ) ;
     Pop (S, x) ;  Push (S, 's' ) ;
```

```
    while (! StackEmpty(S))  { Pop(S, y); printf (y) ; };
    printf (x) ;
}
```

【解答】输出结果：stack

4. 简述以下算法的功能（栈的元素类型SElemType为int）。

```
(1) Status algo1(Stack S) {
        int i,  n,  A[255];
        n=0;
        while (! StackEmpty(S) ) {n++;  Pop(S,  A[n]); };
        for (i=1; i<=n; i++) Push(S,  A[i]);
    }
(2) Status algo2(Stack S,  int e) {
        Stack T;  int d;
        InitStack(T);
        while (! StackEmpty (S)) {
            Pop(S,  d);
            if (d!=e) Push(T,  d);
        }
        while (! StackEmpty(T)) {
            Pop(T,  d);
            Push(S,  d);
        }
    }
```

【分析】本题考查利用栈来人工模拟递归执行过程的掌握。

【解答】（1）将栈S中数据元素次序颠倒。

（2）利用栈T辅助将栈S中所有值为*e*的数据元素删除之。

5. 按照四则运算加、减、乘、除和幂运算（↑）优先关系的惯例，画出对下列算术表达式求值时操作数栈和运算符栈的变化过程：

A-B×C/D+E↑F

【解答】见下表

序号	OPTR 栈	OPND 栈	当前字符												备注（操作）
			A	-	B	*	C	/	D	+	E	^	F	#	
1	#		•												push (OPND, 'A')
2	#	A		•											push (OPTR, '-')
3	#-	A			•										push (OPDN, 'B')
4	#-	AB				•									push (OPTR, '*')
5	#-*	AB					•								push (OPND, 'C')
6	#-*	ABC						•							归约，令T_1=B*C

（续表）

序号	OPTR栈	OPND栈	当前字符												备注（操作）
			A	−	B	*	C	/	D	+	E	^	F	#	
7	#−	AT_1						•							push (OPTR, '/')
8	#−	AT_1							•						push (OPND, 'D')
9	#−/	AT_1D								•					归约，令$T_2=T_1/D$
10	#−/	AT_2								•					归约，令$T_3=A-T_2$
11	#	T_2								•					push (OPTR, '+')
12	#+	T_3									•				push (OPND, 'E')
13	#+	T_3E										•			push(OPTR, '↑')
14	#+↑	T_3E											•		push (OPND, 'F')
15	#+↑	T_3EF												•	归约，令$T_4=E↑F$
16	#+	T_3T_4												•	归约，令$T_5=T_3+T_4$
17	#	T_5												•	return (T_5)

6．试推导求解n阶梵塔问题至少要执行的move操作的次数。

【解答】设至少要执行$M(n)$次move操作，则

$$M(n)=\begin{cases}1 & n=1\\ 2M(n-1)+1 & n>1\end{cases}$$

解此差分方程可求得解为$M(n)=2^n-1$。

7．试将下列递推过程改写为递归过程。

```
void ditui (int n) {
     int i;
     i=n;
     while (i>1)
          printf(i--);
}
```

【分析】由于该递归过程中的递归调用语句出现在过程结束之前，俗称“尾递归”，因此可以不设栈，而通过直接改变过程中的参数值，利用循环结构代替递归调用。

【解答】该递推过程可改写为下列递归过程：

```
void digui (int j)
{
   if (j>1) {
       printf(j);
       digui(j-1);
   }
} //digui
```

8．试将下列递归过程改写为非递归过程。

```
void test(int &sum) {
    int x;
    scanf(x);
    if (x==0) sum=0;
    else {test(sum); sum+=x; }
    printf(sum);
}
```

【分析】该递归过程不能改写成一个简单的递推形式的过程。从它的执行过程可见，其输出的顺序恰好和输入相逆，则必须用一个辅助结构保存其输入值，然后逆向取之，显然用栈最为恰当。

【解答】该递推过程可改写为下列递归过程：

```
void test (int &sum)
{
   Stack S ;
   int x ;
   scanf(x);
   InitStack(S);
   while (x) {
      Push (S,  x);
      scanf(x);
   }
   sum=0;
   printf(sum);
   while (Pop(S, x)) {
       sum += x;
       printf (sum);
   }
}
```

9. 简述队列和栈这两种数据类型的相同点和差异处。

【解答】相同点是它们都是一种插入或删除操作的位置受限的线性表；不同点是栈是后进先出（LIFO）的线性表，而队列是先进先出（FIFO）的线性表。

10. 写出以下程序段的输出结果（队列中的元素类型QElemType为char）。

```
void main( ) {
   Queue Q;  InitQueue(Q);
   char x='e',   y='c';
   EnQueue(Q, 'h');  EnQueue(Q, 'r');  EnQueue(Q, y);
   DeQueue(Q, x);  EnQueue(Q, x);
   DeQueue(Q, x);  EnQueue(Q, 'a');
   while (!QueueEmpty(Q)) {
       DeQueue(Q, y);
       printf(y);
   }
   printf(x);
}
```

【解答】该程序段的输出结果：char。

11. 简述以下算法的功能（栈和队列的元素类型均为int）。

```
void algo3(Queue &Q) {
   Stack S;  int d;
   InitStack (S);
   while (!QueueEmpty(Q)) {
       DeQueue(Q, d);  Push(S, d);
   }
   while (!StackEmpty(S)) {
       Pop(S, d);  EnQueue(Q, d);
   }
}
```

【解答】该算法的功能是利用“栈”作辅助，将“队列”中的数据元素进行逆置。

12. 若以1234作为双端队列的输入序列，试分别求出满足以下条件的输出序列：

（1）能由输入受限的双端队列得到，但不能由输出受限的双端队列得到的输出序列；

（2）能由输出受限的双端队列得到，但不能由输入受限的双端队列得到的输出序列；

（3）既不能由输入受限的双端队列得到，也不能由输出受限的双端队列得到的输出序列。

【解答】（1）4132；（2）4213；（3）4231。

13. 假设以顺序存储结构实现一个双向栈，即在一维数组的存储空间中存在着两个栈，它们的栈底分别设在数组的两个端点。试编写实现这个双向栈tws的3个操作：初始化inistack(tws)、入栈push(tws，i，x)和出栈(tws，i)的算法，其中*i*为0或1，用以分别指示设在数组两端的两个栈，并讨论按过程（正/误状态变量可设为变参）或函数设计这些操作算法各有什么优缺点。

【分析】注意在入栈和出栈的算法中，*i*（=0或1）作为值参出现，因此在算法中应分清两种情况。注意避免易犯的错误，如

```
if (top [0]+top [1]= = m) return OVERFLOW;
else {top [i]=top [i]+1; …}
```

【解答】算法如下：

```
typedef struct{
    Elemtype *base[2];
    Elemtype *top[2];
}BDStacktype; //双向栈类型

Status Init_Stack(BDStacktype &tws, int m)//初始化一个大小为m的双向栈tws
{
    tws.base[0]=(Elemtype*)malloc(sizeof(Elemtype));
    tws.base[1]=tws.base[0]+m;
    tws.top[0]=tws.base[0];
    tws.top[1]=tws.base[1];
```

```
    return OK;
}//Init_Stack

Status push(BDStacktype &tws, int i, Elemtype x)//x入栈，i=0表示低端栈，i=1表示高端栈
{
    if(tws.top[0]>tws.top[1]) return OVERFLOW; //注意此时的栈满条件
    if(i==0) *tws.top[0]++=x;
    else if(i==1) *tws.top[1]--=x;
    else return ERROR;
    return OK;
}//push

Status pop(BDStacktype &tws, int i, Elemtype &x)//x出栈，i=0表示低端栈，i=1表示高端栈
{
    if(i==0)
    {
        if(tws.top[0]==tws.base[0]) return OVERFLOW;
        x=*--tws.top[0];
    }
    else if(i==1)
    {
        if(tws.top[1]==tws.base[1]) return OVERFLOW;
        x=*++tws.top[1];
    }
    else return ERROR;
    return OK;
}//pop
```

14．假设如题1所述火车调度站的入口处有n节硬席或软席车厢（分别以H和S表示）等待调度，试编写算法，输出对这n节车厢进行调度的操作（即入栈或出栈操作）序列，以使所有的软席车厢都被调整到硬席车厢之前。

【分析】注意两侧的铁道均为单向行驶道，且两侧不相通。所有车辆都必须通过“栈道”进行调度。

【解答】算法如下：

```
void Train_arrange(char *train)//这里用字符串train表示火车，'H'表示硬席，'S'表示软席
{
    p=train;q=train;
    InitStack(s);
    while(*p)
    {
        if(*p=='H') push(s, *p); //把'H'存入栈中
        else *(q++)=*p; //把'S'调到前部
        p++;
    }
    while(!StackEmpty(s))
    {
        pop(s, c);
        *(q++)=c; //把'H'接在后部
```

```
    }
}//Train_arrange
```

15. 试写一个算法，识别依次读入的一个以@为结束符的字符序列是否为形如‘序列$_1$&序列$_2$’模式的字符序列。其中序列$_1$和序列$_2$中都不含字符‘&’，且序列$_2$是序列$_1$的逆序列。例如，‘a+b&b+a’是属该模式的字符序列，而‘1+3&3-1’则不是。

【分析】在写算法之前，应先分析清楚不符合给定模式的几种情况。注意题中给出的条件：模式串中应含字符‘&’，则不含字符‘&’的字符序列与模式串也不匹配。

【解答】算法如下：

```
int IsReverse()//判断输入的字符串中'&'前和'&'后部分是否为逆串，是则返回1，否则返回0
{
    InitStack(s);
    while((e=getchar())!='&')
         push(s, e);
    while((e=getchar())!='@')
    {
        if(StackEmpty(s)) return 0;
        pop(s, c);
        if(e!=c) return 0;
    }
    if(!StackEmpty(s)) return 0;
    return 1;
}//IsReverse
```

16. 试写一个判别表达式中开、闭括号是否配对出现的算法。

【分析】由于表达式中只含一种括弧，只有两种错误情况，即：在没有左括弧的情况下，出现右括弧或者整个表达式中的左括弧数目多于右括弧。因此只需要附设一个计数器，记录已经出现的、且尚未有右括弧与之配对的左括弧的个数。

【解答】算法如下：

```
Status Bracket_Test(char *str)//判别表达式中左右括号是否匹配
{
    count=0;
    for(p=str;*p;p++)
    {
        if(*p=='(') count++;
        else if(*p==')') count--;
        if (count<0) return ERROR;
    }
    if(count) return ERROR; //注意括号不匹配的两种情况
    return OK;
}//Bracket_Test
```

17. 假设一个算术表达式中可以包含3种括号：圆括号“（”和“）”、方括号“[”和“]”和花括号“{”和“}”，且这3种括号可按任意的次序嵌套使用（如：…[…{…}…[…]…]…[…]…(…)…）。编写判别给定表达式中所含括号是否正确配对出现的算法（已

知表达式已存入数据元素为字符的顺序表中）。

【分析】和16题不同，这是一个需要借助于栈来处理的典型问题，它具有天然的后进先出特性。可按“期待匹配消解”的思想来设计算法，对表达式中每一个左括弧都“期待”一个相应的右括弧与之匹配，且自左至右按表达式中出现的先后论，越迟的左括弧期待匹配的渴望程度越高。反之，不是期待出现的右括弧则是非法的。

【解答】算法如下：

```
Status AllBrackets_Test(char *str)//判别表达式中3种括号是否匹配
{
    InitStack(s);
    for(p=str;*p;p++)
    {
        if(*p=='(' || *p=='[' || *p=='{') push(s, *p);
        else if(*p==')' || *p==']' || *p=='}')
        {
            if(StackEmpty(s)) return ERROR;
            pop(s, c);
            if(*p==')'&&c!='(') return ERROR;
            if(*p==']'&&c!='[') return ERROR;
            if(*p=='}'&&c!='{') return ERROR; //必须与当前栈顶括号匹配
        }
    }//for
    if(!StackEmpty(s)) return ERROR;
    return OK;
}//AllBrackets_Test
```

18. 假设以二维数组$g(1..m, 1..n)$表示一个图像区域，$g[i, j]$表示该区域中点（i，j）所具颜色，其值为从0到k的整数。编写算法置换点（i_0，j_0）所在区域的颜色。约定和（i_0，j_0）同色的上、下、左、右的邻接点为同色区域的点。

【分析】本算法采用类似于图的广度优先遍历的思想，用两个队列保存相邻同色点的横坐标和纵坐标。

【解答】算法如下：

```
typedef struct {
    int x;
    int y;
} coordinate;
void Repaint_Color(int g[m][n], int i, int j, int color)//把点(i, j)相邻区域的颜色置换为color
{
    old=g[i][j];
    InitQueue(Q);
    EnQueue(Q, {i, j});
    while(!QueueEmpty(Q))
    {
        DeQueue(Q, a);
        x=a.x;y=a.y;
```

```
        if(x>1)
              if(g[x-1][y]==old)
              {
                  g[x-1][y]=color;
                  EnQueue(Q, {x-1, y}); //修改左邻点的颜色
          }
          if(y>1)
              if(g[x][y-1]==old)
              {
                  g[x][y-1]=color;
                  g[x][y-1]=color;
                  EnQueue(Q, {x, y-1}); //修改上邻点的颜色
         }
         if(x<m)
              if(g[x+1][y]==old)
              {
                  g[x+1][y]=color;
                  EnQueue(Q, {x+1, y}); //修改右邻点的颜色
              }
              if(y<n)
          if(g[x][y+1]==old)
          {
                g[x][y+1]=color;
                EnQueue(Q, {x, y+1}); //修改下邻点的颜色
          }
     }//while
}//Repaint_Color
```

19. 假设表达式由单字母变量和双目四则运算符构成。试写一个算法，将一个通常书写形式且书写正确的表达式转换为逆波兰式。

【分析】注意所有的变量在逆波兰式中出现的先后顺序和在原表达式中出现的相同，因此只需要设立一个“栈”，根据操作符的“优先级”调整它们在逆波兰式中出现的顺序。

【解答】算法如下：

```
void NiBoLan(char *str, char *new)//把中缀表达式str转换成逆波兰式new
{
    p=str;q=new; //为方便起见，设str的两端都加上了优先级最低的特殊符号
    InitStack(s); //s为运算符栈
    while(*p)
    {
        if(*p是字母) *q++=*p; //直接输出
        else
        {
            c=gettop(s);
            if(*p优先级比c高) push(s, *p);
            else
            {
                 while(gettop(s)优先级不比*p低)
                 {
```

```
                pop(s, c);*(q++)=c;
              }//while
              push(s, *p); //运算符在栈内遵循越往栈顶优先级越高的原则
            }//else
          }//else
          p++;
      }//while
}//NiBoLan
```

20. 如题19的假设条件，试写一个算法，对以逆波兰式表示的表达式求值。

【解答】算法如下：

```
int GetValue_NiBoLan(char *str)//对逆波兰式求值
{
  p=str;InitStack(s); //s为操作数栈
  while(*p)
  {
    if(*p是数) push(s, *p);
    else
    {
      pop(s, a);pop(s, b);
      r=compute(b, *p, a); //假设compute为执行双目运算的过程
      push(s, r);
    }//else
    p++;
  }//while
  pop(s, r);return r;
}//GetValue_NiBoLan
```

21. 如题19的假设条件，试写一个算法，判断给定的非空后缀表达式是否为正确的逆波兰式（即后缀表达式），如果是，则将它转化为波兰式（即前缀表达式）。

【分析】若以顺序表表示后缀表达式，即以每个分量为一个字符的一维数组存储表达式，则设一个数据元素为字符串的栈，以存放在分析后缀表达式过程中得到的子前缀表达式；若以单链表（每个结点的数据域存放一个字符）表示表达式，则每个栈元素为指示构成子前缀表达式的第一个结点和最后一个结点的一对指针。在以顺序表表示表达式时，也可以设一个数据元素为字符的栈，则在转化过程中得到的子表达式可表示为栈中的一个字符序列，并在两个子表达式之间加一分隔符。顺序扫描表达式，若当前字符是变量，则该字符就是一个子前缀表达式；若当前字符是运算符O，则它和栈顶元素S_2、次栈顶元素S_1构成一个新的子前缀表达式（OS_1S_2）。子前缀表达式可用C语言中的串表示。正确性的判断可在转换过程中进行。

【解答】算法如下：

```
Status NiBoLan_to_BoLan(char *str, stringtype &new)//把逆波兰表达式str转换为波兰式new
{
  p=str;Initstack(s); //s的元素为stringtype类型
  while(*p)
  {
```

```
        if(*p为字母) push(s, *p);
        else
        {
          if(StackEmpty(s)) return ERROR;
          pop(s, a);
          if(StackEmpty(s)) return ERROR;
          pop(s, b);
          c=link(link(*p, b), a);
          push(s, c);
        }//else
        p++;
      }//while
      pop(s, new);
      if(!StackEmpty(s)) return ERROR;
      return OK;
    }//NiBoLan_to_BoLan
```

22. 试编写如下定义的递归函数的递归算法，并根据算法画出求$g(5, 2)$时栈的变化过程。

$$g(m, n)=\begin{cases}0 & m=0,\ n\geqslant 0\\ g(m-1, 2n)+n & m>0,\ n\geqslant 0\end{cases}$$

【解答】算法如下：

```
Status g(int m, int n, int &s)//求递归函数g的值s
{
    if(m==0&&n>=0) s=0;
    else if(m>0&&n>=0) s=n+g(m-1, 2*n);
         else return ERROR;
    return OK;
}//g
```

23. 试写出求递归函数$F(n)$的递归算法，并消除递归：

$$F(n)=\begin{cases}n+1 & n=0\\ n\cdot F(n/2) & n>0\end{cases}$$

【解答】算法如下：

```
Status F_recursive(int n, int &s)//递归算法
{
    if(n<0) return ERROR;
    if(n==0) s=n+1;
    else
    {
        F_recurve(n/2, r);
        s=n*r;
    }
```

```
    return OK;
}//F_recursive
Status F_nonrecursive(int n, int s)//非递归算法
{
    if(n<0) return ERROR;
    if(n==0) s=n+1;
    else
    {
        InitStack(s); //s的元素类型为struct {int a;int b;}
        while(n!=0)
        {
            a=n;b=n/2;
            push(s, {a, b});
            n=b;
        }//while
        s=1;
        while(!StackEmpty(s))
        {
            pop(s, t);
            s*=t.a;
        }//while
    }//else
    return OK;
}//F_nonrecursive
```

24．求解平方根$\sqrt{A}$的迭代函数定义如下：

$$\text{sqrt}(A,P,e)=\begin{cases}P & |P^2-A|<e\\ \text{sqrt}(A,\frac{1}{2}(P+\frac{A}{P}),e) & |P^2-A|\geq e\end{cases}$$

其中，p是A的近似平方根，e是结果允许误差。试写出相应的递归算法，并消除递归。

【分析】通过本题，掌握迭代法，迭代本身便是用递归形式来表达，本质是逐步降低问题规模。

【解答】算法如下：

```
float Sqrt_recursive(float A, float p, float e)//求平方根的递归算法
{
    if(abs(p^2-A)<=e) return p;
    else return sqrt_recurve(A, (p+A/p)/2, e);
}//Sqrt_recurve

float Sqrt_nonrecursive(float A, float p, float e)//求平方根的非递归算法
{
    while(abs(p^2-A)>=e)
        p=(p+A/p)/2;
    return p;
}//Sqrt_nonrecursive
```

25．已知Ackerman函数的定义如下：

$$\mathrm{akm}(m,\ n)=\begin{cases} n+1 & m=0 \\ \mathrm{akm}(m-1,\ 1) & m\neq 0,\ n=0 \\ \mathrm{akm}(m-1,\ \mathrm{akm}(m,\ n-1)) & m\neq 0,\ n\neq 0 \end{cases}$$

（1）写出递归算法；

（2）写出非递归算法。

【解答】（1）递归算法如下：

```
int akm (int m,  int n)
{
  if (m = = 0) akm=n+1;
  else if (n = = 0)
        akm=akm (m-1,  1);
     else {
        g=akm (m,  n-1);
        akm=akm (m-1,  g);
     }
} //akm
```

（2）非递归算法如下：

```
int akm1 (int m,  int n)
{// S[MAX]为附设栈，top为栈顶指针
    top=0; S[top].mval=m; S[top].nval=n;
    do {
        while (S[top].mval) {
            while (S[top].nval) {
                  top ++; S[top].mval=S[top-1].mval;
                  S[top].nval=S[top-1].nval-1;
            }
            S[top].mval--; S[top].nval=1;
        }
        if (top>0) {
             top--;
             S[top].mval--;
             S[top].nval=S[top+1].nval+1;
        }
    } while (top! = 0  | |  S[top].mval! = 0);
    akm1=S[top].nval+1;  top--;
} //akm1
```

26. 假设以带头结点的循环链表表示队列，并且只设一个指针指向队尾元素结点（注意不设头指针），试编写相应的队列初始化、入队列和出队列的算法。

【分析】此题注意出队列操作在队列中只有一个元素时的特殊情形需单独处理。

【解答】算法如下：

```
void InitCiQueue(CiQueue &Q)//初始化循环链表表示的队列Q
{
```

```
  Q=(CiLNode*)malloc(sizeof(CiLNode));
  Q->next=Q;
}//InitCiQueue

void EnCiQueue(CiQueue &Q, int x)// 把元素x插入循环链表表示的队列Q,Q指向队尾元素,Q->next
                                 // 指向头结点, Q->next->next指向队头元素
{
  p=(CiLNode*)malloc(sizeof(CiLNode));
  p->data=x;
  p->next=Q->next; //直接把p加在Q的后面
  Q->next=p;
  Q=p;  //修改尾指针
}

Status DeCiQueue(CiQueue &Q, int x)//从循环链表表示的队列Q头部删除元素x
{
  if(Q==Q->next) return INFEASIBLE; //队列已空
  p=Q->next->next;
  x=p->data;
  Q->next->next=p->next;
  free(p);
  return OK;
}//DeCiQueue
```

27. 如果希望循环队列中的元素都能得到利用，则需设置一个标志域tag，并以tag的值为0或1来区分，尾指针和头指针值相同时的队列状态是“空”还是“满”。试编写与此结构相应的入队列和出队列的算法，并从时间和空间角度讨论设标志和不设标志这两种方法的使用范围（如当循环队列容量较小而队列中每个元素占的空间较多时，哪一种方法较好）。

【分析】标志位tag的初值置“0”。一旦元素入队列使rear == front时，需置tag为“1”；反之，一旦元素出队列使front == rear时，需置tag为“0”，以便使下一次进行入队列或出队列操作时（此时front == rear），可由标志位tag的值来区别队列当时的状态是“满”，还是“空”。

【解答】算法如下：

```
Status EnCyQueue(CyQueue &Q, int x)//带tag域的循环队列入队算法
{
    if(Q.front==Q.rear&&Q.tag==1) //tag域的值为0表示"空", 1表示"满"
        return OVERFLOW;
    Q.base[Q.rear]=x;
    Q.rear=(Q.rear+1)%MAXSIZE;
    if(Q.front==Q.rear) Q.tag=1; //队列满
}//EnCyQueue

Status DeCyQueue(CyQueue &Q, int &x)//带tag域的循环队列出队算法
{
    if(Q.front==Q.rear&&Q.tag==0) return INFEASIBLE;
```

```
    Q.front=(Q.front+1)%MAXSIZE;
    x=Q.base[Q.front];
    if(Q.front==Q.rear) Q.tag=1; //队列空
    return OK;
}//DeCyQueue
```

28. 假设将循环队列定义为：以域变量rear和length分别指示循环队列中队尾元素的位置和内含元素的个数。试给出此循环队列的队满条件，并写出相应的入队列和出队列的算法（在出队列的算法中要返回队头元素）。

【分析】设head指示循环队列中队头元素的位置，则有

head= ((q.rear+MAXLEN) - q.length+1) % MAXLEN

其中，MAXLEN为队列可用的最大空间。

【解答】算法如下：

```
Status EnCyQueue(CyQueue &Q, int x)//带length域的循环队列入队算法
{
    if(Q.length==MAXLEN) return OVERFLOW;
    Q.rear=(Q.rear+1)%MAXLEN;
    Q.base[Q.rear]=x;
    Q.length++;
    return OK;
}//EnCyQueue

Status DeCyQueue(CyQueue &Q, int &x)//带length域的循环队列出队算法
{
    if(Q.length==0) return INFEASIBLE;
    head=(Q.rear-Q.length+1)%MAXLEN;
    x=Q.base[head];
    x=Q.base[head];
    Q.length--;
}//DeCyQueue
```

29. 假设称正读和反读都相同的字符序列为“回文”，例如，'abba'和'abcba'是回文，'abcde'和'ababab'则不是回文。试写一个算法判别读入的一个以'@'为结束符的字符序列是否是“回文”。

【分析】由于依次输入的字符序列中不含特殊的分隔符，则在判别是否是“回文”时，一种比较合适的做法是，同时利用“栈”和“队列”两种结构。

【解答】算法如下：

```
int Palindrome_Test()//判别输入的字符串是否为回文序列，是则返回1，否则返回0
{
    InitStack(S);InitQueue(Q);
    while(c=getchar()!='@')
    {
        Push(S, c);EnQueue(Q, c); //同时使用栈和队列两种结构
    }
    while(!StackEmpty(S))
```

```
    {
        Pop(S, a);DeQueue(Q, b);
        if(a!=b) return ERROR;
    }
    return OK;
}//Palindrome_Test
```

30. 试利用循环队列编写求k阶斐波那契序列中前n+1项（f_0，f_1，…，f_n）的算法，要求满足：f_n≤max而f_{n+1}>max，其中max为某个约定的常数。（注意：本题所用循环队列的容量仅为k，则在算法执行结束时，留在循环队列中的元素应是所求k阶斐波那契序列中的最后k项f_{n-k+1}，…，f_n）。

【分析】由于循环队列中只有k个元素空间，则在计算f_i(i≥k)时，队列总是处在头尾相接的状态，故仅需一个指针rear指示当前队尾的位置。每次求得一个f_i之后即送入(rear+1)% k的位置上，冲掉原队头元素。若将题目条件减弱为允许含k+1个分量的数组，则在计算时可利用简化公式。

【解答】算法如下：

```
void GetFib_CyQueue(int k, int n)//求k阶斐波那契序列的前n+1项
{
    InitCyQueue(Q); //其MAXSIZE设置为k
    for(i=0;i<k-1;i++) Q.base[i]=0;
    Q.base[k-1]=1; //给前k项赋初值
    for(i=0;i<k;i++) printf("%d", Q.base[i]);
    for(i=k;i<=n;i++)
    {
        m=i%k;sum=0;
        for(j=0;j<k;j++) sum+=Q.base[(m+j)%k];
        Q.base[m]=sum; //求第i项的值存入队列中并取代已无用的第一项
        printf("%d", sum);
    }
}//GetFib_CyQueue
```

31. 在顺序存储结构上实现输出受限的双端循环队列的入列和出列（只允许队头出列）算法。设每个元素表示一个待处理的作业，元素值表示作业的预计时间。入队列采取简化的短作业优先原则，若一个新提交的作业的预计执行时间小于队头和队尾作业的平均时间，则插入在队头，否则插入在队尾。

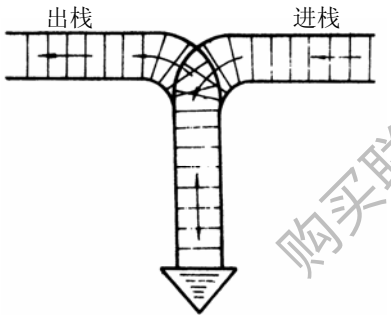
【分析】① 注意队列满和队列为空的判别条件，②注意对于插入位置的确定，通过一个作业平均时间判定（短作业优先原则在操作系统中有着广泛的应用）。

【解答】算法如下：

```
Status EnDQueue(DQueue &Q, int x)//输出受限的双端队列的入队操作
{
    if((Q.rear+1)%MAXSIZE==Q.front) return OVERFLOW; //队列满
    avr=(Q.base[Q.rear-1]+Q.base[Q.front])/2;
    if(x>=avr) //根据x的值决定插入在队头还是队尾
    {
```

```
        Q.base[Q.rear]=x;
        Q.rear=(Q.rear+1)%MAXSIZE;
    } //插入在队尾
    else
    {
        Q.front=(Q.front-1)%MAXSIZE;
        Q.base[Q.front]=x;
    } //插入在队头
    return OK;
}//EnDQueue
Status DeDQueue(DQueue &Q, int &x)//输出受限的双端队列的出队操作
{
    if(Q.front==Q.rear) return INFEASIBLE; //队列空
    x=Q.base[Q.front];
    Q.front=(Q.front+1)%MAXSIZE;
    return OK;
}//DeDQueue
```

32．假设在下图所示的铁道转轨网的输入端有n节车厢：硬座、硬卧和软卧（分别以P，H和S表示）等待调度，要求这三种车厢在输出端铁道上的排列次序为：硬座在前，软卧在中，硬卧在后。试利用输出受限的双端队列对这n节车厢进行调度，编写算法输出调度的操作序列：分别以字符'E'和'D'表示对双端队列的头端进行入队列和出队列的操作；以字符A表示对双端队列的尾端进行入队列的操作。



【分析】注意对于P、S、H三种车厢的不同处置。

【解答】算法如下：

```
void Train_Rearrange(char *train)// 这里用字符串train表示火车，'P'表示硬座，'H'表示硬卧，'S'表示软
                                 // 卧，最终按PSH的顺序排列
{
    r=train;
    InitDQueue(Q);
    while(*r)
    {
        if(*r=='P')
        {
            printf("E");
            printf("D"); //实际上等于不入队列，直接输出P车厢
```

```
        }
        else if(*r=='S')
        {
            printf("E");
            EnDQueue(Q, *r, 0); //0表示把S车厢从头端入队列
        }
        else
        {
            printf("A");
            EnDQueue(Q, *r, 1); //1表示把H车厢从尾端入队列
        }
    }//while
    while(!DQueueEmpty(Q))
    {
        printf("D");
        DeDQueue(Q);
    }//while //从头端出队列的车厢必然是先S后H的顺序
}//Train_Rearrange
```

3.4　考研真题分析

例题3-1　（清华大学1998年试题）

双端队列（Deque）是一种可以在任何一端进行插入和删除的线性表，现采用一个一维数组作为双端队列的数据存储结构，使用类Pascal语言描述如下：

```
CONST maxsize=32
TYPE deque=RECORD
     elem:ARRAY [0..maxsize-1]OF datatype;  {环形队列的存放数组}
     end1,   end2:0..maxsize;                    {环形队列的两端}
end;
```

试编写两个算法add(Qu:deque;x:datatype;tag:0..1)和delete(Qu:deque;var x:data-type; tag:0..1)，用以在此双端队列的任一端进行插入和删除。当tag=0时，在左端end1端操作，当tag=1时，在右端end2端操作。

【解答】算法如下：

```
FUNC add(Qu:deque;x:datatype;tag:0..1):integer;
{双端队列的初始状态（空队列）为Qu.end1+1=Qu.end2；双端队列满的条件为Qu.end1=Qu.end2;如果插入成功，则返回插入数据在数组中的下标，否则返回-1}
CASE tag OF
     0:BEGIN
          IF(Qu.end1 !=Qu.end2)THEN
          BEGIN
                    Qu.elem[Qu.end1]:=x;
                    Qu.end1:=(Qu.end1-1)MOD maxsize;
                    return((Qu.end1+1)MOD maxsize);
```

```
            END
            ELSE return(-1);
        END
        1:BEGIN
            IF(Qu.end2!=Qu.end1)THEN
            BEGIN
                Qu.elem[Qu.end2]:=x;
                Qu.end2:=(Qu.end2+1)MOD maxsize;
                return((Qu.end2-1)MOD maxsize);
            END
            ELSE return(-1);
          END

ENDC;{add}
ENDF;{add}

FUNC delete(Qu:deque, VAR x:dataype, tag:0..1):integer;
{如果删除成功则返回0, 否则返回-1}
CASE tag OF
    0:BEGIN
            IF(((Qu.end1+1)MOD maxsize)!=Qu.end2)THEN
            BEGIN
                i:=(Qu.end1+1)MOD maxsize;
                WHILE(i!=Qu.end2)AND(Qu.elem[Qu[i]]!=x)DO
                    i:=(i+1)MOD maxsize;
                IF(Qu.elem[i]=x AND i!=Qu.end2)THEN
                BEGIN
                    j:=i;
                    WHILE((j-1)MOD maxsize!=Qu.end1)DO
                    BEGIN
                        Qu.elem[j]:=Qu.elem[(j-1)MOD maxsize];
                        j:=(j-1)MOD maxsize;
                    END
                    Qu.end1:=(Qu.end1+1)MOD maxsize;
                    return(0);
                END
                ELSE return(-1);
            END
            ELSE return(-1);
    END
    1:BEGIN
        IF(((Qu.end2-1)MOD maxsize)!=Qu.end1)THEN
        BEGIN
            i:=(Qu.end2-1)MOD maxsize;
            WHILE(i!=Qu.end1)AND(Qu.elem[Qu[i]]!=x)DO
                i:=(i-1)MOD maxsize;
            IF(Qu.elem[i]=x AND i!=Qu.end1)THEN
            BEGIN
                j:=i;
                WHILE((j+1)MOD maxsize!=Qu.end2)DO
```

```
                BEGIN
                    Qu.elem[j]:=Qu.elem[(j+1)MOD maxsize];
                    j:=(j+1)MOD maxsize;
                END
                Qu.end1:=(Qu.end2-1)MOD maxsize;
                return(0);
            END
            ELSE return(-1);
        END{if}
        ELSE return(-1);
    END{case 1}
    ENDC;
ENDF; {delete}
```

例题3-2 （清华大学1998年试题）

用单链表表示的链式队列的队头在链表的_____ 位置。

A. 链头　　B. 链尾　　C. 链中

【分析】队列的队头是对队列元素进行删除的一端。链队列的队头在链表的链头位置。（不考虑不包含数据元素的头结点。）

【解答】A

例题3-3 （中国科学院计算技术研究院1999年试题）

设一单向链表的头指针为head，链表的记录中包含整数类型的key域，试设计算法，将此链表的记录按照key递增的顺序进行就地排序。

【解答】 算法如下：

```
TYPE pointer=^chaintype;
    chaintype=RECORD
    key:integer;
    next:pointer;
END

PROC sort(head:pointer)
{head 为指向链表头结点的指针；对链表进行插入排序}
        IF(head^.next=NIL)THEN r:=NIL;
        ELSE r:=head^.next^.next;
        WHILE (r<>NIL) DO
        BEGIN
            m:=r^.key;
            p:=head;
            q:=head^.next;
            WHILE (m>q^.key) DO
            BEGIN
                p:=q;
                p:=q^.next;
            END
            IF (q<>r) THEN
```

```
                BEGIN
                    t:=r;
                    r:=r^.next;
                    t^.next:=q;
                    p^.next:=t;
                END
                ELSE
                    r:=r^.next;
            END
ENDP; {sort}
```

例题3-4　（中国科学院软件研究所1999年试题）

判断正误：

栈和队列都是限制存取点的线性结构。

【解答】　正确

例题3-5　（中国科学技术大学1998年试题）

消除递归不一定需要使用栈，此说法（　　）。

A. 正确　　B. 错误

【分析】对于尾递归可以将其转换成递推，不需要栈。

【解答】 A

例题3-6　（北京邮电大学1998年试题）

判断正误：

有n个数顺序（依次）进栈，出栈序列有C_n种。

$$C_n=\frac{1}{n+1}\times\frac{(2n)!}{(n!)^2}$$

【分析】假设对二叉树的n个结点从1到n加以编号，且令其前序序列为1，2，…，n，则不同的二叉树所得的中序序列不同。因此，不同形态的二叉树的数目恰好是前序序列均为1，2，…，n的二叉树所能得到的中序序列的数目，而中序遍历的过程实质上是一个结点进栈和出栈的过程。因此，数列1，2，…，n按不同顺序进栈和出栈所能得到的排列的数目恰好为由前序序列1，2，…，n所能得到的中序序列的数目。

这个数目为$C_n=\frac{1}{n+1}\times\frac{(2n)!}{(n!)^2}$。

【解答】正确

例题3-7　（清华大学1999年试题）

一个双端队列Deque是限定在两端，end1和end2都可以进行插入和删除的线性表。队空条件是end1=end2。若用顺序方式来组织双端队列，试根据下列要求，定义双端队列的结构，并给出在指定端i(i=1，2)进行插入enq和删除deq操作的实现。

要求：

① 当队满时，最多只能有一个元素空间可以是空的。
② 在进行两端的插入和删除时，队列中其他元素是一律不动。

【解答】操作实现如下：

```
FUNC add(Qu:Deque;x:datatype;tag:0..1):integer;
BEGIN
     CASE tag OF
     0:BEGIN
          IF(Qu.end1<>Qu.end2)THEN
          BEGIN
               Qu.elem[Qu.end1]:=x;
               Qu.end1:=(Qu.end1-1)MOD maxsize;
               RETURN((Qu.end1+1)MOD maxsize);
          END
          ELSE return(-1);
     END

     1:BEGIN
          IF(Qu.end2<>Qu.end1)THEN
          BEGIN
               Qu.elem[Qu.end2]:=x;
               Qu.end2:=(Qu.end2+1)MOD maxsize;
               return((Qu.end2-1)MOD maxsize);
          END
          ELSE return(-1);
     END

     ENDCASE;{add}
ENDF;{add}

FUNC delete(Qu:Deque;VAR x:datatype;tag:0..1):integer;
BEGIN
     CASE tag OF
     0:BEGIN
          IF(((Qu.end1+1)MOD maxsize)<>Qu.end2)THEN
          BEGIN
               i:=(Qu.end1+1)MOD maxsize;
               WHILE(i<>Qu.end2)AND(Qu.elem[Qu[i]]<>x)DO
                    i:=(i+1)MOD maxsize;
               IF(Qu.elem[i]=x AND i<>Qu.end2)THEN
               BEGIN
                    j:=i;
                    WHILE((j-1)MOD maxsize<>Qu.end1)DO
                    BEGIN
                         Qu.elem[j]:=Qu.elem[(j-1)MOD maxsize];
                         j:=(j-1)MOD maxsize;
                    END
                         Qu.end1:=(Qu.end1+1)MOD maxsize;
                         return(0);
```

```
                END
                ELSE return(-1);
            END
            ELSE return(-1);
        END
        1:BEGIN
            IF(((Qu.end2-1)MOD maxsize)<>Qu.end1)THEN
            BEGIN
                i:=(Qu.end2-1)MOD maxsize;
                WHILE(I<>Qu.end1)AND(Qu.elem[Qu[i]]<>x)DO
                    i:=(i-1)MOD maxsize;
                IF(Qu.elem[i]=x AND i<>Qu.end1)THEN
                BEGIN
                    j:=i;
                    WHILE((j+1)MOD maxsize<>Qu.end2)DO
                    BEGIN
                        Qu.elem[j]:=Qu.elem[(j+1)MOD maxsize];
                        j:=(j+1)MOD maxsize;
                    END
                    Qu.end1:=(Qu.end2-1)MOD maxsize;
                    return(0);
                END
                ELSE return(-1);
            END{if}
            ELSE return(-1);
        END{case 1}
        ENDCASE;
    END;{delete}
```

例题3-8　（清华大学2002年试题）

判断下列叙述的正误：

栈、先进先出队列、优先级队列、双端队列等都可以看作是一个容器类的派生类。该容器代表限制存取位置的顺序存取结构。

【解答】正确

例题3-9　（中国科学院计算机网络信息中心2000年试题）

算法题：

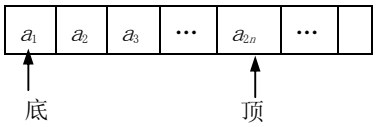
设顺序栈S中有$2n$个元素，从栈顶到栈底的元素依次是a_{2n}，a_{2n-1}，…，a_2，a_1，要求通过一个辅助的循环队列及相应的入栈、出栈、入队、出队操作来重新排列栈中元素，使得从栈顶到栈底的元素依次是a_{2n}，a_{2n-2}，…，a_4，a_2，a_{2n-1}，a_{2n-3}，…，a_3，a_1，请写出一算法实现该操作，要求附加的空间是$O(n)$，时间复杂度为$O(n)$。

【解答】算法如下：

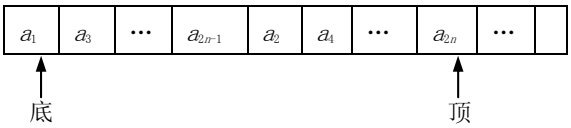
```
PROC sort (s:sqstktp);
BEGIN
    INIQVEVE(Q);
```

```
        while not Empty (s) do
             ENQVEVE(Q, pop(s));
        i:=2n;
        while(i>=0) do
        BEGIN
             if (i mod 2=0) then ENQVEVE(Q, DLQVEVE(Q))
             else push(s, DLQVEVE(Q));
             i:=i-1;
        END
        while not Empty (s) do
             ENQVEVE(Q, pop(s));
        i:=0;
        while(i<=n) do
        BEGIN
             push(s, DLQVEVE(Q));
             i:=i+1;
        END
        while not Empty(s) do
             ENQVEVE(Q, pop(s));
        while not Empty(Q) do
             push(s, DLQVEVE(Q));
END
```

①所有元素出栈入队列；②偶数结点出队列再入队列，与此同时奇数结点出队列入栈；③奇数结点出栈入队列（即倒向）；④偶数结点出队列入栈，再出栈入队列（即倒向）；⑤所存元素出队列且入栈即为所求。



栈的原始状态



栈的最终状态

例题3-10　（中国科学院计算机网络信息中心2000年试题）

设栈的输入序列是1，2，3，4，则_____不可能是其出栈序列。

A. 1，2，4，3　　B. 2，1，3，4　　C. 1，4，3，2

D. 4，3，1，2　　E. 3，2，1，4

【解答】D

例题3-11　（中国人民大学2002年试题）

简述如何对一个前缀算术表达式求值的算法。

【解答】建一个操作数栈，从前辍表达式的末端往前取操作数或运算符，若取到的是操作数则入栈，若取的是运算符则从栈中弹出两个操作数进行该运算，并将运算结果入栈，直到整个表达式扫描完。

例题3-12 （北京科技大学2002年试题）

请简述在你所进行的算法设计中运用到栈和队列的两个例子。

【解答】 BFS和DFS。

例题3-13 （武汉理工大学2002年试题）

简答题：

A、B、C三个元素进栈s的次序是A、B、C，利用Push(s，x)，Pop(s)表示入栈、出栈操作，写出所有可能的出栈序列和获得每个序列的相应操作，并指明哪个序列不会是出栈序列。

【解答】

CBA push(S，A) push (S，B) push(S，C) pop(S) pop(S) pop(S)

BCA push(S，A) push (S，B) pop(S) push(S，C) pop(S) pop(S)

ACB push(S，A) pop(S) push (S，B) push(S，C) pop(S) pop(S)

ABC push(S，A) pop(S) push (S，B) pop(S) push(S，C) pop(S)

BAC push(S，A) push (S，B) pop(S) pop(S) push(S，C) pop(S)

不可能的出栈序列为CAB。

例题3-14 （南京理工大学2002年试题）

简答题：

在操作序列push(1)，push(2)，pop，push(5)，push(7)，pop，push(6)之后，栈顶元素和栈底元素分别是什么？

push(k)表示整数k入栈，pop表示栈顶元素出栈。

【解答】 栈顶元素为6，栈底元素为1。

例题3-15 （南京理工大学2002年试题）

简答题：

在操作序列Qinsert(1)，Qinsert(2)，Qdelete，Qinsert(5)，Qinsert(7)，Qdelete，Qinsert(9)之后，队头元素和队尾元素分别是什么？

Qinsert(k)表示整数k入队列，Qdelete表示元素出队列。

【解答】 队头元素为5，队尾元素为9。

例题3-16 （南京理工大学2002年试题）

选择题：

循环队列A[0..m-1]存放其元素，用front和rear分别表示队头及队尾，则循环队列满的条件是____。

A. (Q.rear+1)%m==Q.front　　B. Q.rear==Q.front+1

C. Q.rear+1==Q.front　　D. Q.rear==Q.front

【解答】A

例题3-17 （南京理工大学2002年试题）

下面的算法是由终端输入集合A和B元素，建立静态链表S，计算集合A-B。

```
void difference(slinklist &space, int &s){
       Initspace_sl(space);
       s=Malloc_sl(space);
       r=s;
       scanf(m, n);
       for(j=1;j<=m;++j){
           i=Malloc_sl(space);
           scanf(space[i].data);
           space[r].cur=i;    (1)    ;
       }//for
          (2)    ;
       for(j=1;i<=n;++j){
             scanf(b);p=s;    (3)    ;
             while(k)!=space[r].cur &&    (4)    {
                   p=k;    (5)    ;
             }//while
       if(k!=space[r].cur){
           space[p].cur=space[k].cur;
           free_sl(space, k);
       }//if
           if(r==k)    (6)    ;
           }//for
}//difference
```

注：

Initspace_sl(Slinklist &space)：将一维数组space中各分量链成一个备用链，space[0].cur为头指针。

Malloc_sl(Slinklist &space)：分配结点，若备用链非空，返回分配结点的下标，否则返回0。

Free_sl(Slinklist &space, int k)：将下标为k的空闲结点回收到备用链。

【解答】

（1）r=i

（2）space[r].cur=0

（3）k=space[s].cur

（4）space[k].data!=b

（5）k=space[k].cur

（6）r=p

3.5 自 测 题

自测题3-1

一般情况下，将递归算法转换成等价的非递归算法应该设置（　　）。

A. 堆栈　　B. 队列　　C. 堆栈和队列　　D. 数组

自测题3-2

中缀表达式A-(B+C/D)*E的后缀形式是（　　）。

A. AB-C+D/E*　　B. ABC+D/-E*

C. ABCD/E*+-　　D. ABCD/+E*-

自测题3-3

试给出运算变量均为整数的简单算术表达式所需最少临时单元个数的算法（假定不许用代数规则变更表达式的计值顺序）。

例如，A+B*C需要一个临时单元为(B*C)；(A+B)*(C+D)+E则需要两个临时单元，一个为C+D，另一个既为(A+B)又为(A+B)*(C+D)。

自测题3-4

在解决计算机主机与打印机之间速度不匹配问题时通常设置一个打印数据缓冲区，主机将要输出的数据依次写入该缓冲区，而打印机则从该缓冲区中取出数据打印。该缓冲区应该是一个_____结构。

A. 堆栈　　B. 队列　　C. 数组　　D. 线性表

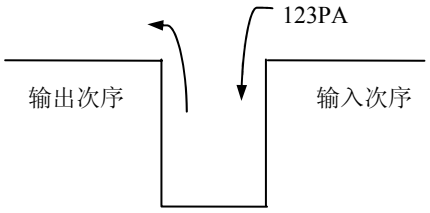
自测题3-5

写出和下列递归过程等价的非递归过程。

```
PROCEDURE test(VAR sum:integer);
    VAR a:integer;
BEGIN
    read(a);
    IF a=0 THEN sum:=1
    ELSE BEGIN
        test(sum);
        sum:=sum*a
    END;
    write(sum)
END;
```

自测题3-6

设输入元素为1、2、3、P和A，输入次序为123PA，元素经过栈后到达输出序列，当所有元素均到达输出序列后，有哪些序列可以作为高级语言的变量名？



3.6　自测题答案

【自测题3-1】堆栈的用途之一就是将递归转化为递归。

答案为A。

【自测题3-2】

答案为D。

【自测题3-3】

设运算变量均为整数的某个简单算术表达式，已加工为k个四元式，n_1是起始四元式号码，n_k是终止四元式号码。由于四元式序列是由加工简单算术表达式得到的，因此这些四元式均为一目或二目算术运算的四元式。为使存入中间结果的临时单元个数最少，我们设立一个计数器count。当四元式的第2或第3项出现临时变量（即出现对临时变量的引用时），每出现一个，计数器应减少1；当四元式的第4项（即存放运算结果的项）出现临时变量时，计数器就应增加1。

正如前面所述，这些四元式是加工简单算术表达式得到的算术运算四元式，因此每个四元式必然要把计算结果赋予某个临时变量，因此四元式的第4项一定是临时变量。

于是对每个算术运算四元式来说，若四元式第2和第3项都是临时变量时，计数器要减少1；当第2、第3项只有一个是临时变量时，计数器不变；当第2与第3项都不是临时变量时，计数器增加1。

下面给出的算法顺便也给出了临时变量分配的临时单元地址（假定当前分配的临时单元的起始地址为A）。

算法如下：

```
PROCEDURE  PT(a, n1, nk);
BEGIN
```

```
        count:=0;
        p:=0;              /*p是临时单元个数 */
        FOR i:=n1  to nk DO
        BEGIN
             IF 四元式的第2，3项都是临时变量  THEN
                count:=count-1;
             IF 四元式的第2，3项都不是临时变量  THEN
             BEGIN
                  count:=count+1;
                  p:=max(p，count);
                  T[i].ADDR:=a+count-1          /*保存起始地址为a */
             END
        END
END
```

其中p记录这k个四元式所需临时单元的最大个数；count是计数器。例如，假设有一个整形的算术表达式为A*B-C*D+E*F，生成的四元式以及所需最少临时单元数如下所示。

$$\begin{pmatrix} \text{四元式} & \text{地址} & count & p \\ (*,A,B,T_1) & a & 1 & 1 \\ (*,C,D,T_2) & a+1 & 2 & 2 \\ (-,T_1,T_2,T_3) & a & 1 & 2 \\ (*,E,F,T_4) & a+1 & 2 & 2 \\ (+,T_3,T_4,T_5) & a & 1 & 2 \end{pmatrix}$$

【自测题3-4】

答案为B。

【自测题3-5】

```
PROCEDURE test;
VAR stack:  array[1..MAXSIZE]of INTEGER;
     top，sum，a:INTEGER;
BEGIN
     top:=0;
     sum:=1;
     read(a);
     WHILE a<>0 DO
     BEGIN
          top:=top+1;
          stack[top]:=a;
          read(a);
     END
     write(sum);
     WHILE top<>0 DO
     BEGIN
          sum:=sum*stack[top];
          top:=top-1;
          write(sum);
```

```
    END
END
```

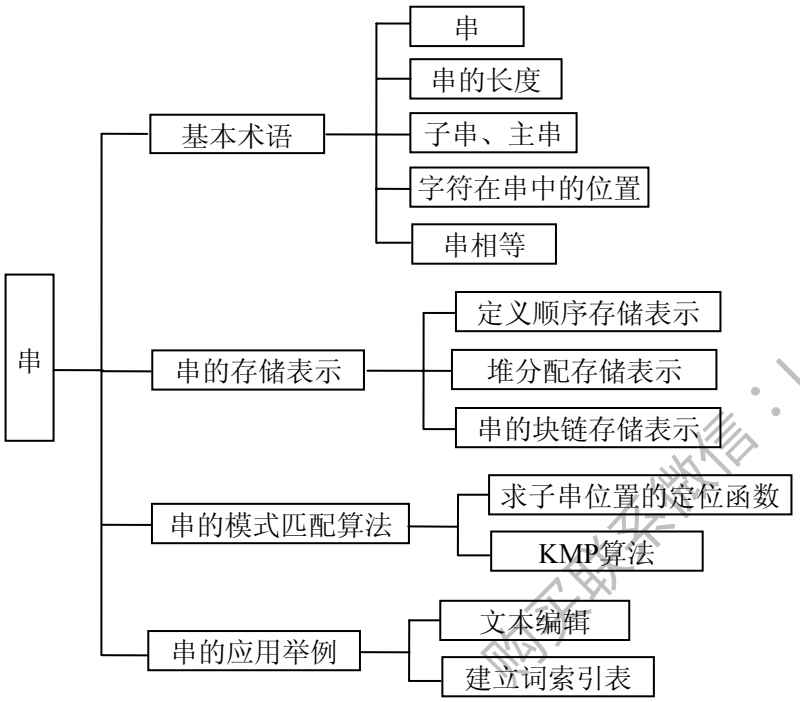
【自测题3-6】

高级语言变量名首字符为字母。

答案：AP321，PA321，P3A21，P32A1，P321A。

第 4 章　串

4.1　基本知识结构图



4.2　知 识 点

1. 基本概念

串是由零个或多个字符组成的有限序列，记为$s=$'$a_1a_2\cdots a_n$'（$n\geqslant 0$）。其中s是串的名，用单引号括起来的字符序列是串的值，串中字符的数目n称为串的长度。

长度为0的串称为空串。

串中任意个连续的字符组成的子序列称为该串的子串。

包含子串的串相应地称为主串。

通常称字符在序列中的序号为该字符在串中的位置。子串在主串中的位置以子串的第

一个字符在串中的位置来表示。

当且仅当两个串的值相等时称这两个串是相等的。

2. 串的存储结构

串的存储结构有定长顺序存储表示、堆分配存储表示和块链存储表示。

定长顺序存储表示：用一组地址连续的存储单元存储串值的字符序列。对串长的表示方法有两种：以下标为0的数组分量存放串的实际长度和在串值后面加一个不计入串长的结束标记字符。定长顺序存储表示法在实现中如果出现串值序列的长度超过上界时需用结尾法处理，为克服这个弊端只有采用动态分配串值的存储空间。

堆分配存储表示：以一组地址连续的存储单元存放串值字符序列，但它们的存储空间是在程序执行过程中动态分配的。该方法既有定长顺序存储结构的特点，处理方便，操作中对串长又没有任何限制。

块链存储表示：采用链表方式存储串值，每个结点可以存放一个字符，也可以存放多个字符。

3. 串的模式匹配算法

串的模式匹配是串的各种操作中最重要的一种。KMP算法是一种比较高效的模式匹配算法，尤其当模式与主串之间存在许多“部分匹配”时，KMP算法比一般的匹配算法快得多，而且在该算法中指示主串的指针不需回溯。

4.3　习题及参考答案

1. 简述空串和空格串（或称空格符串）的区别。

【解答】空格串是由一个或多个空格组成的串，它的长度为串中空格字符的个数；空串用符号“∅”表示，它的长度为0。

2. 设s='I AM A STUDENT'，t='GOOD'，q='WORKER'。

求：StrLength(s)，　StrLength(t)，　SubString(s，8，7)，　SubString(t，2，1)，

Index(s，'A')，　Index(s，t)，　Replace(s，'STUDENT'，q)，

Concat(SubString(s，6，2)，Concat(t，SubString(s，7，8)))。

【解答】StrLength(s) = 14

StrLength(t) = 4

SubString(s，8，7) = 'STUDENT'

SubString(t，2，1) = '0'

Index(s，'A') = 3

Index(s，t) = 0

Replace(s，'STUDENT'，q) = 'I AM A WORKER'

Concat(SubString(s, 6, 2), Concat(t, SubString(s, 7, 8))) = 'A GOOD STUDENT'

3. 试问执行以下函数会产生怎样的输出结果?

```
void demonstrate ( ) {
    StrAssign(s, 'THIS IS A BOOK');
    Replace(s, SubString(s, 3, 7), 'ESE ARE');
    StrAssign(t, Concat(s, 'S'));
    StrAssign(u, 'XYXYXYXYXYXY');
    StrAssign(v, SubString(u, 6, 3));
    StrAssign(w, 'W');
    printf('t=', t, 'v=', v, 'u=', Replace(u, v, w));
} //demonstrate
```

【解答】t='THESE ARE BOOKS', v='YXY', u='XWXWXW'。

4. 令s='aaab', t='abcabaa', u='abcaabbabcabaacbacba'。试分别求出它们的next函数值和nextval函数值。

【分析】注意模式匹配算法中关键的一步是求next函数。

【解答】

j	1	2	3	4
串s	a	a	a	b
next [j]	0	1	2	3
nextval [j]	0	0	0	3

j	1	2	3	4	5	6	7
串t	a	b	c	a	b	a	a
next [j]	0	1	1	1	2	3	2
nextval [j]	0	1	1	0	1	3	2

j	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
串u	a	b	c	a	a	b	b	a	b	c	a	b	a	a	c	b	a	c	b	a
next [j]	0	1	1	1	2	2	3	1	2	3	4	5	3	2	2	1	1	2	1	1
nextval [j]	0	1	1	0	2	1	3	0	1	1	0	5	3	2	2	1	0	2	1	0

5. 在以链表存储串值时，存储密度是结点大小和串长的函数。假设每个字符占一个字节，每个指针占4个字节，每个结点的大小为4的整数倍。已知串长的分布函数为$f(l)$且$\sum_{l=0}^{\max len} f(l)$，求结点大小为$4k$，串长为$l$时的存储密度$d(4k, l)$（用公式表示）。

【分析】考查串的链表存储结构特点及存储密度定义。

【解答】$d(4k,l)=\dfrac{l}{4(k+1)\cdot\left\lfloor\dfrac{l}{4k}\right\rfloor}$

编写6~10题的算法时，请用StringType数据类型：

StringType是串的一个抽象数据类型，它包含以下五种基本操作：

```
void StrAssign(StringType& t, StringType s)
//将s的值赋给t。s的实际参数可以是串变量或者串常量（如：'abcd'）。
int StrCompare(StringType s, StringType t)
//比较s和t。若s>t，返回值>0；若s=t，返回值=0；若s<t，返回值<0。
int StrLength(StringType s)
//返回s中的元素个数，即该串的长度。
StringType Concat(StringType s, StringType t)
//返回由s和t联接而成的新串。
StringType SubString(StringType s, int start, int len)
//当1≤start≤StrLength(s)且0≤len≤StrLength(s)-start+1时，
//返回s中第start个字符起长度为len的子串，否则返回空串。
```

6．编写对串求逆的递推算法。

【分析】从串尾起依次取单个字符添加到新串中，考查灵活运用串的操作。

【解答】算法如下：

```
void String_Reverse(StringType s, StringType &r)//求s的逆串r
{
   StrAssign(r, ''); //初始化r为空串
   for(i=Strlen(s);i;i--)
   {
       StrAssign(c, SubString(s, i, 1));
       StrAssign(r, Concat(r, c)); //把s的字符从后往前添加到r中
   }
}//String_Reverse
```

7．编写算法，求得所有包含在串s中而不包含在串t中的字符（s中重复的字符只选一个）构成的新串r，以及r中每个字符在s中第一次出现的位置。

【解答】算法如下：

```
void String_Subtract(StringType s, StringType t, StringType &r)//求所有包含在串s中而t中没有的字符
                                                              //构成的新串r
{
    StrAssign(r, '');
    for(i=1;i<=Strlen(s);i++)
    {
        StrAssign(c, SubString(s, i, 1));
        for(j=1;j<i&&StrCompare(c, SubString(s, j, 1));j++); //判断s的当前字符c是否第一次出现
```

```
        if(i==j)
        {
            for(k=1;k<=Strlen(t)&&StrCompare(c, SubString(t, k, 1));k++);
                                                    //判断当前字符是否包含在t中
            if(k>Strlen(t)) StrAssign(r, Concat(r, c));
        }
    }//for
}//String_Subtract
```

8. 编写一个实现串的置换操作Replace(&S, T, V)的算法。

【解答】算法如下：

```
int Replace(StringType &S, StringType T, StringType V);//将串S中所有子串T替换为V, 并返回置
                                                    //换次数
{
  for(n=0, i=1;i<=Strlen(S)-Strlen(T)+1;i++) //注意i的取值范围
    if(!StrCompare(SubString(S, i, Strlen(T)), T)) //找到了与T匹配的子串
    { //分别把T的前面和后面部分保存为head和tail
      StrAssign(head, SubString(S, 1, i-1));
      StrAssign(tail, SubString(S, i+Strlen(T), Strlen(S)-i-Strlen(T)+1));
      StrAssign(S, Concat(head, V));
      StrAssign(S, Concat(S, tail)); //把head, V, tail连接为新串
      i+=Strlen(V); //当前指针跳到插入串以后
      n++;
    }//if
  return n;
}//Replace
```

9. 编写算法，从串*s*中删除所有和串*t*相同的子串。

【解答】算法如下：

```
int Delete_SubString(StringType &s, StringType t)//从串s中删除所有与t相同的子串, 并返回删除次数
{
  for(n=0, i=1;i<=Strlen(s)-Strlen(t)+1;i++)
    if(!StrCompare(SubString(s, i, Strlen(t)), t))
    {
      StrAssign(head, SubString(S, 1, i-1));
      StrAssign(tail, SubString(S, i+Strlen(t), Strlen(s)-i-Strlen(t)+1));
      StrAssign(S, Concat(head, tail)); //把head, tail连接为新串
      n++;
    }//if
  return n;
}//Delete_SubString
```

10. 利用串的基本操作以及栈和集合的基本操作，编写“由一个算术表达式的前缀式求后缀式”的递推算法（假设前缀式不含语法错误）。

【分析】将前缀串中各个字符入栈，再利用栈的先进后出特性得到后缀。

【解答】算法如下：

```
Status NiBoLan_to_BoLan(StringType str, StringType &new)//把前缀表达式str转换为后缀式new
{
    Initstack(s); //s的元素为Stringtype类型
    for(i=1;i<=Strlen(str);i++)
    {
        r=SubString(str, i, 1);
        if(r为字母) push(s, r);
        else
        {
            if(StackEmpty(s)) return ERROR;
            pop(s, a);
            if(StackEmpty(s)) return ERROR;
            pop(s, b);
            StrAssign(t, Concat(r, b));
            StrAssign(c, Concat(t, a)); //把算符r, 子前缀表达式a, b连接为新子前缀表达式
c
            push(s, c);
        }
    }//for
    pop(s, new);
    if(!StackEmpty(s)) return ERROR;
    return OK;
}//NiBoLan_to_BoLan
```

在编写11~14题的算法时，请采用定长顺序存储表示法，而不允许调用串的基本操作。

11. 编写算法，实现串的基本操作StrAssign(&T, chars)。

【解答】算法如下：

```
void StrAssign(StringType &T, char chars&#;)//用字符数组chars给串T赋值
{
    for(i=0, T[0]=0;chars[i];T[0]++, i++) T[i+1]=chars[i];
}//StrAssign
```

12. 编写算法，实现串的基本操作StrCompare(S, T)。

【解答】算法如下：

```
char StrCompare(StringType s, StringType t)//串的比较, s>t时返回正数, s=t时返回0, s<t时
返回负数
{
  for(i=1;i<=s[0]&&i<=t[0]&&s[i]==t[i];i++);
  if(i>s[0]&&i>t[0]) return 0;
  else if(i>s[0]) return -t[i];
     else if(i>t[0]) return s[i];
         else return s[i]-t[i];
}//StrCompare
```

13. 编写算法，实现串的基本操作Replace(&S, T, V)。

【分析】置换过程中应注意避免数组越界，可将算法设计成函数，若置换过程中引起数组越界，则置函数值为FALSE。

【解答】算法如下：

```
bool repl (SString &NewS,  SString S,  SString T,  SString V )
{
    //以串V置换串S中所有和T相同的子串后构成一个新串NewS。
    //若因置换引起数组越界，则返回FALSE，否则返回TRUE。
    m=S[0]; n=T[0]; overflow=FALSE;
    i=k=1; l=0;
    //i指示串S中当前待进行置换的剩余串的起始位置，
    //l指示串NewS的当前长度。
    while (!overflow && m-k+1>=n )
    {
        j=1;
        while (j <= n &&S[k+j-1] = = T[j]) j++;
        if (j <= n) k++;
        else   //匹配成功
           if (l+k-i+V[0]>MAXSTRLEN ) overflow = TRUE;
           else {
                    copy (NewS,  S,  l+1,  i,  k-i);
                    copy (NewS,  V,  l+k-i, 1, V[0]);
                    i=k+n; k=i; l=l+k-i+V[0];
            }
      }
      if (!overflow && i <=m &&&+(m-i+1)<=MAXSTRLEN)
        copy (NewS,  S, l+1,  i, m-i+1);
      else overflow = TRUE;
      return (! overflow);
} //repl
```

注意 上述算法中过程copy (a，b，x，y，len)的功能，是将串b中自第y个字符起共len个字符，复制到串a中第x个字符起的位置上。

14. 编写算法，求串*s*所含不同字符的总数和每种字符的个数。

【分析】设置一个类似第9题的串*r*（初始为空串），rlen对*r*的当前长度计数，数组Count[length(s)]（初值为0）依次对工作串*r*中每个字符在*s*中出现的次数计数。当扫描到串*s*的字符s[i]时：如果s[i]不在*r*中，则将其接到*r*的末尾，rlen加1，Count[rlen-1]置为1。如果s[i]已在r中，假设s[i]=r[k]，则Count[k-1]加1。

【解答】算法如下：

```
typedef struct {
                char ch;
                int num;
            } mytype;
void StrAnalyze(Stringtype S)//统计串S中字符的种类和个数
{
    mytype T[MAXSIZE]; //用结构数组T存储统计结果
```

```
    for(i=1;i<=S[0];i++)
    {
        c=S[i];j=0;
        while(T[j].ch&&T[j].ch!=c) j++; //查找当前字符c是否已记录过
        if(T[j].ch) T[j].num++;
        else T[j]={c, 1};
    }//for
    for(j=0;T[j].ch;j++)
      printf("%c:  %d\n", T[j].ch, T[j].num);
}//StrAnalyze
```

15. 假设以结点大小为1（且附设头结点）的链表结构表示串。试编写实现下列六种串的基本操作StrAssign，StrCopy，StrCompare，StrLength，Concat和SubString的函数。

【分析】按照题目的要求（各函数之间可以自由地嵌套），在实现上述操作时必须做到：（1）结果串与自变量串不共享空间；（2）操作不破坏自变量串；（3）基本操作要求高效执行，彼此之间不能调用。同时为了便于区分合法串与非法串，建议合法串一律用带头结点的单链表表示，以空指针表示非法串，当自变量中有一个非法串时，函数值可为-1或INVALID或NULL。

【解答】算法如下：

```
typedef struct{
                char ch;
                LStrNode *next;
              } LStrNode, *LString; //链串结构
void StringAssign(LString &s, LString t)//把串t赋值给串s
{
    s=malloc(sizeof(LStrNode));
    for(q=s, p=t->next;p;p=p->next)
    {
        r=(LStrNode*)malloc(sizeof(LStrNode));
        r->ch=p->ch;
        q->next=r;q=r;
    }
    q->next=NULL;
}//StringAssign
void StringCopy(LString &s, LString t)//把串t复制为串s。与前一个程序的区别在于，串s已存在
{
    for(p=s->next, q=t->next;p&&q;p=p->next, q=q->next)
    {
        p->ch=q->ch;pre=p;
    }
    while(q)
    {
        p=(LStrNode*)malloc(sizeof(LStrNode));
        p->ch=q->ch;
        pre->next=p;pre=p;
    }
```

```
    p->next=NULL;
}//StringCopy
char StringCompare(LString s, LString t)//串的比较，s>t时返回正数，s=t时返回0，s<t时返回负数
{
    for(p=s->next, q=t->next;p&&q&&p->ch==q->ch;p=p->next, q=q->next);
    if(!p&&!q) return 0;
    else if(!p) return -(q->ch);
        else if(!q) return p->ch;
              else return p->ch-q->ch;
}//StringCompare
int StringLen(LString s)//求串s的长度(元素个数)
{
    for(i=0, p=s->next;p;p=p->next, i++);
    return i;
}//StringLen
LString * Concat(LString s, LString t)//连接串s和串t形成新串，并返回指针
{
    p=malloc(sizeof(LStrNode));
    for(q=p, r=s->next;r;r=r->next)
    {
        q->next=(LStrNode*)malloc(sizeof(LStrNode));
        q=q->next;
        q->ch=r->ch;
    }//for //复制串s
    for(r=t->next;r;r=r->next)
    {
        q->next=(LStrNode*)malloc(sizeof(LStrNode));
        q=q->next;
        q->ch=r->ch;
    }//for //复制串t
    q->next=NULL;
    return p;
}//Concat
LString * Sub_String(LString s, int start, int len)//返回一个串，其值等于串s从start位置起长为len的
                                                    //子串
{
    p=malloc(sizeof(LStrNode));q=p;
    for(r=s;start;start--, r=r->next);  //找到start所对应的结点指针r
    for(i=1;i<=len;i++, r=r->next)
    {
        q->next=(LstrNode*)malloc(sizeof(LStrNode));
        q=q->next;
        q->ch=r->ch;
    } //复制串t
    q->next=NULL;
    return p;
}//Sub_String
```

16．假设以块链结构表示串。试编写将串s插入到串t中某个字符之后的算法（若串t中不存在此字符，则将串s联接在串t的末尾）。

【分析】为了节省插入或删除的运算时间，应避免字符在结点间大量移动，而应利用非串字符（如@）来填补结点中多余空间。

【解答】算法如下：

```
void LString_Concat(LString &t, LString &s, char c)//用块链存储结构，把串s插入到串t的字符c之后
{
    p=t.head;
    while(p&&!(i=Find_Char(p, c))) p=p->next; //查找字符c
    if(!p) //没找到
    {
        t.tail->next=s.head;
        t.tail=s.tail; //把s连接在t的后面
    }
    else
    {
        q=p->next;
        r=(Chunk*)malloc(sizeof(Chunk)); //将包含字符c的结点p分裂为两个
        for(j=0;j<i;j++) r->ch[j]='#'; //原结点p包含c及其以前的部分
        for(j=i;j<CHUNKSIZE;j++) //新结点r包含c以后的部分
        {
            r->ch[j]=p->ch[j];
            p->ch[j]='#'; //p的后半部分和r的前半部分的字符改为无效字符'#'
        }
        p->next=s.head;
        s.tail->next=r;
        r->next=q; //把串s插入到结点p和r之间
    }//else
    t.curlen+=s.curlen; //修改串长
    s.curlen=0;
}//LString_Concat
int Find_Char(Chunk *p, char c)//在某个块中查找字符c，如找到则返回位置是第几个字符，如没找到
                               //则返回0
{
    for(i=0;i<CHUNKSIZE&&p->ch[i]!=c;i++);
    if(i==CHUNKSIZE) return 0;
    else return i+1;
}//Find_Char
```

17．假设以块链结构作串的存储结构。试编写判别给定串是否具有对称性的算法，并要求算法的时间复杂度为O(StrLength(S))。

【解答】算法如下：

```
int LString_Palindrome(LString L)//判断以块链结构存储的串L是否为回文序列，是则返回1，否则返回0
```

```
{
    InitStack(S);
    p=S.head;i=0;k=1; //i指示元素在块中的下标，k指示元素在整个序列中的序号(从1开始)
    for(k=1;k<=S.curlen;k++)
    {
        if(k<=S.curlen/2) Push(S, p->ch[i]); //将前半段的字符入串
        else if(k>(S.curlen+1)/2)
        {
            Pop(S, c); //将后半段的字符与栈中的元素相匹配
            if(p->ch[i]!=c) return 0; //失配
        }
        if(++i==CHUNKSIZE) //转到下一个元素，当为块中最后一个元素时，转到下一块
        {
             p=p->next;
             i=0;
        }
    }//for
    return 1; //成功匹配
}//LString_Palindrome
```

在编写18~20题的算法时，请采用堆分配存储表示。

18. 试写一算法，在串的堆存储结构上实现串基本操作Concat(&T, s1, s2)。

【解答】算法如下：

```
void HString_Concat(HString s1, HString s2, HString &t)//将堆结构表示的串s1和s2连接为新串t
{
    if(t.ch) free(t.ch);
    t.ch=malloc((s1.length+s2.length)*sizeof(char));
    for(i=1;i<=s1.length;i++) t.ch[i-1]=s1.ch[i-1];
    for(j=1;j<=s2.length;j++, i++) t.ch[i-1]=s2.ch[j-1];
    t.length=s1.length+s2.length;
}//HString_Concat
```

19. 试写一算法，实现堆存储结构的串的置换操作Replace(&S, T, V)。

【解答】算法如下：

```
int HString_Replace(HString &S, HString T, HString V)//堆结构串上的置换操作，返回置换次数
{
    for(n=0, i=0;i<=S.length-T.length;i++)
    {
        for(j=i, k=0;k<T.length&&S.ch[j]==T.ch[k];j++, k++);
        if(k==T.length) //找到了与T匹配的子串:分三种情况处理
        {
            if(T.length==V.length)
               for(l=1;l<=T.length;l++) //新子串长度与原子串相同时:直接替换
                   S.ch[i+l-1]=V.ch[l-1];
            else if(T.length<V.length) //新子串长度大于原子串时:先将后部右移
```

```
            {
                  for(l=S.length-1;l>=i+T.length;l--)
                        S.ch[l+V.length-T.length]=S.ch[l];
                  for(l=0;l<V.length;l++)
                        S[i+l]=V[l];
            }
            else //新子串长度小于原子串时:先将后部左移
            {
                  for(l=i+V.length;l<S.length+V.length-T.length;l++)
                        S.ch[l]=S.ch[l-V.length+T.length];
                  for(l=0;l<V.length;l++)
                        S[i+l]=V[l];
             }
          S.length+=V.length-T.length;
          i+=V.length;n++;
        }//if
    }//for
    return n;
}//HString_Replace
```

20. 试写一算法，实现堆存储结构的串的插入操作StrInsert(&S，pos，T)。

【解答】算法如下：

```
Status HString_Insert(HString &S, int pos, HString T)//把T插入堆结构表示的串S的第pos个
字符之前
{
  if(pos<1) return ERROR;
  if(pos>S.length) pos=S.length+1;//当插入位置大于串长时，看作添加在串尾
  S.ch=realloc(S.ch, (S.length+T.length)*sizeof(char));
  for(i=S.length-1;i>=pos-1;i--)
    S.ch[i+T.length]=S.ch[i]; //后移为插入字符串让出位置
  for(i=0;i<T.length;i++)
    S.ch[pos+i-1]=T.ch[pos]; //插入串T
  S.length+=T.length;
  return OK;
}//HString_Insert
```

21. 当以定长顺序结构表示串时，可如下所述改进定位函数的算法：先将模式串t中的第一个字符和最后一个字符与主串s中相应的字符比较，在两次比较都相等之后，再依次从t的第二个字符起逐个比较。这样做可以克服算法Index在求模式串'a^kb'（a^k表示连续k个字符'a'）在主串'a^nb'($k\leqslant n$)中的定位函数时产生的弊病。试编写上述改进算法，并比较这两种算法在作Index('a^nb'，'a^kb')运算时所需进行的字符间的比较次数。

【分析】当$k\leqslant n$时，原算法的比较次数为(k+1) (n-k+1)，改进算法的比较次数为2(n-k)+ (k+1)。

【解答】算法如下：

```
int Index_New(StringType s, StringType t)//改进的定位算法
{
```

```
    i=1;j=1;
    while(i<=s[0]&&j<=t[0])
    {
        if((j!=1&&s[i]==t[j]) || (j==1&&s[i]==t[j]&&s[i+t[0]-1]==t[t[0]]))
        { //当j==1即匹配模式串的第一个字符时，需同时匹配其最后一个
          i=i+j-2;
          j=1;
        }
        else
        {
            i++;j++;
        }
    }//while
    if(j>t[0]) return i-t[0];
}//Index_New
```

22. 假设以结点大小为1（带头结点）的链表结构表示串，则在利用next函数值进行串匹配时，在每个结点中需设三个域：数据域chdata，指针域succ和指针域next。其中chdata域存放一个字符；succ域存放指向同一链表中后继结点的指针；next域在主串中存放指向同一链表中前驱结点的指针；在模式串中，存放指向当该结点的字符与主串中的字符不等时，模式串中下一个应进行比较的字符结点（即与该结点字符的next函数值相对应的字符结点）的指针，若该结点字符的next函数值为零，则其next域的值应指向头结点。试按上述定义的结构改写求模式串的next函数值的算法。

【解答】算法如下：

```
void LGet_next(LString &T)//链串上的get_next算法
{
    p=T->succ;p->next=T;q=T;
    while(p->succ)
    {
        if(q==T || p->data==q->data)
        {
            p=p->succ;q=q->succ;
            p->next=q;
        }
        else q=q->next;
    }//while
}//LGet_next
```

23. 试按22题定义的结构改写串匹配的改进算法（KMP算法）。

【解答】算法如下：

```
LStrNode * LIndex_KMP(LString S, LString T, LStrNode *pos)//链串上的KMP匹配算法，返回值为匹
                                                          //配的子串首指针
{
    p=pos;q=T->succ;
    while(p&&q)
```

```
    {
        if(q==T || p->chdata==q->chdata)
        {
            p=p->succ;
            q=q->succ;
        }
        else q=q->next;
    }//while
    if(!q)
    {
        for(i=1;i<=Strlen(T);i++)
            p=p->next;
        return p;
    } //发现匹配后，要往回找子串的头
    return NULL;
}//LIndex_KMP
```

24. 假设以定长顺序存储结构表示串，试设计一个算法，求串s中出现的第一个最长重复子串及其位置，并分析你的算法的时间复杂度。

【分析】该算法的思想是，依次把串s的一个副本s2向右错位平移1格，2格，3格，……与自身s1相匹配，如果存在最长重复子串，则必然在此过程中被发现。用变量lrs1、lrs2、maxlen来记录已发现的最长重复子串第一次出现的位置、第二次出现的位置和长度。题目中未说明“重复子串”是否允许有重叠部分，本算法假定允许。如不允许，只需在第二个for语句的循环条件中加上k<=i即可。此题可有多种解法，但时间复杂度不同，较好的做法能达到的时间复杂度为$O(\text{LENGTH}^2(s))$。

【解答】算法如下：

```
void Get_LRepSub(StringType s)//求s的最长重复子串的位置和长度
{
    for(maxlen=0, i=1;i<s[0];i++)//串s2向右移i格
    {
        for(k=0, j=1;j<=s[0]-i;j++)//j为串s2的当前指针，此时串s1的当前指针为i+j，两指针同步移动
        {
            if(s[j]==s[j+i]) k++;     //用k记录连续相同的字符数
            else k=0;            //失配时k归零
            if(k>maxlen)              //发现了比以前发现的更长的重复子串
            {
                lrs1=j-k+1;lrs2=mrs1+i;maxlen=k; //作记录
            }
        }//for
    }//for
    if(maxlen)
    {
        printf("Longest Repeating Substring length:%d\n", maxlen);
        printf("Position1:%d  Position 2:%d\n", lrs1, lrs2);
    }
    else printf("No Repeating Substring found!\n");
```

```
}//Get_LRepSub
```

25．假设以定长顺序存储结构表示串，试设计一个算法，求串*s*和串*t*的一个最长公共子串，并分析你的算法的时间复杂度。若要求第一个出现的最长公共子串（即它在串*s*和串*t*的最左边的位置上出现）和所有的最长公共子串，讨论你的算法能否实现。

【分析】本题基本思路与上题同。惟一的区别是，由于A，B互不相同，因此B不仅要向右错位，而且还要向左错位，以保证不漏掉一些情况。当B相对于A的位置不同时，需要匹配的区间的计算公式也各不相同，请读者自己画图以帮助理解。本算法的时间复杂度是*O*(strlrn(*s*)*strlen(*t*))。

【解答】算法如下：

```
void Get_LPubSub(StringType S, StringType T)//求串S和串T的最长公共子串位置和长度
{
    if(S[0]>=T[0])
    {
        StrAssign(A, S);StrAssign(B, T);
    }
    else
    {
        StrAssign(A, T);StrAssign(B, S);
    } //为简化设计，令S和T中较长的那个为A，较短的那个为B
    for(maxlen=0, i=1-B[0];i<A[0];i++)
    {
        if(i<0) //i为B相对于A的错位值，向左为负，左端对齐为0，向右为正
        {
            jmin=1;jmax=i+B[0];
        }//B有一部分在A左端的左边
        else if(i>A[0]-B[0])
       {
           jmin=i;jmax=A[0];
       }//B有一部分在A右端的右边
       else
       {
           jmin=i;jmax=i+B[0];
       }//B在A左右两端之间
//以上是根据A和B不同的相对位置确定A上需要匹配的区间(与B重合的区间)的端点:jmin, jmax
        for(k=0, j=jmin;j<=jmax;j++)
        {
            if(A[j]==B[j-i]) k++;
            else k=0;
            if(k>maxlen)
            {
                lps1=j-k+1;lps2=j-i-k+1;maxlen=k;
            }
        }//for
    }//for
    if(maxlen)
    {
```

```
        if(S[0]>=T[0])
        {
            lpsS=lps1;lpsT=lps2;
        }
        else
        {
              lpsS=lps2;lpsT=lps1;
        } //将A，B上的位置映射回S，T上的位置
        printf("Longest Public Substring length:%d\n", maxlen);
        printf("Position in S:%d  Position in T:%d\n", lpsS, lpsT);
    }//if
    else printf("No Repeating Substring found!\n");
}//Get_LPubSub
```

4.4　考研真题分析

例题4-1　（清华大学1998年试题）

设目标为t="abcaabbabcabaacbacba"，模式为p="abcabaa"，

① 计算模式p的nextval函数值；

② 不写出算法，只画出利用KMP算法进行模式匹配时每一趟的匹配过程。

【解答】① p的nextval函数值见下表。

j	1	2	3	4	5	6	7
模式串	a	b	c	A	b	a	a
nextval[j]	0	1	1	0	0	3	0

② 利用KMP算法进行模式匹配时每一趟的匹配过程如下所示。

第一趟匹配：

abcaabbabcabaacbacba

abcab

第二趟匹配：

abcaabbabcabaacbacba

abc

第三趟匹配：

abcaabbabcabaacbacba

a

第四趟匹配：

abcaabbabcabaacbacba

abcabaa

第四趟匹配成功。

例题4-2 （清华大学1998年试题）

设目标为S="abcaabbcaaababababaabca"，模式为P="babab"，

① 手工计算P的nextval值；

② 写出利用求得的nextval数组，按KMP算法对目标S进行模式匹配的过程。

【解答】① P的nextval值见下表。

J	1	2	3	4	5
模式	b	a	b	a	b
nextval[j]	0	1	0	1	0

② 按KMP算法对目标S进行模式匹配的过程如下所示。

第一趟匹配：

abcaabbcaaababababaabca

b

第二趟匹配：

abcaabbcaaababababaabca

ba

第三趟匹配：

abcaabbcaaababababaabca

b

第四趟匹配：

abcaabbcaaababababaabca

b

第五趟匹配：

abcaabbcaaababababaabca

b

第六趟匹配：

abcaabbcaaababababaabca

ba

第七趟匹配：

abcaabbcaaababababaabca

ba

第八趟匹配：

abcaabbcaaababababaabca

b

第九趟匹配：

abcaabbcaaababababaabca
b
第十趟匹配：
abcaabbcaaababababaabca
b
第十一趟匹配：
abcaabbcaaababababaabca
b
第十二趟匹配：
abcaabbcaaababababaabca
babab

匹配成功。

例题4-3　（东北大学1998年试题）

已知3个字符串分别为s='ab…abcaabcbca…a'，s'='caab'，s″='bcb'，
利用所学字符串基本运算的函数得到结果串为：
s‴'='caabcbca…aca…a'。
要求写出得到上述结果串s所用的函数及执行算法。

【解答】算法如下：

```
FUNC catstr(var s,  s1, s2, s3: string):string;
    var i1, i2:integer;
       t1, t2:string;
BEGIN
      i1:=index(s, s1, 1);
      i2:=index(s, s2, 1)+3;
      t1:=substring(s, i1, strlength(s)-i1+1);
      t2:=substring(s, i2, strlength(s)-i2+1);
      s3:=concat(t1, t2);
      return(s3);
ENDF
```

例题4-4　（北京邮电大学1998年试题）

模式串t='abcaabbcabcaabdab'，该模式串的next数组的值为_____，nextval数组的值为_____。

供选择的答案：

A. 01112211123456712
B. 01112121123456112
C. 01110013101100701
D. 01112231123456712
E. 01100111011001701

F. 01102131011021701

【解答】该模式串的next数组的值为01112231123456712，即选择D。
nextval数组的值为01102131011021701，即选择F。

4.5 自 测 题

自测题4-1

字符串S满足下式，其中Head和Tail的定义同广义表类似，如Head('xyz')='x'，Tail('xyz')='yz'，则S=_____。

Concat(Head(Tail(S))，Head(Tail(Tail)(S)))='dc'

A. abcd　　B. acbd　　C. acdb　　D. adcb

自测题4-2

设S为一个长度为n的字符串，其中的字符各不相同，则S中的互异的非平凡子串（非空且不同于S本身）的个数为_____。

A. 2^{n-1}　　B. n^2　　C. $(n^2/2)+(n/2)$

D. $(n^2/2)+(n/2)-1$　　E. $(n^2/2)-(n/2)-1$　　F. 其他情况

自测题4-3

假设串的存储结构如下所示，编写算法实现串的置换操作。

```
TYPE strtp=RECORD
        Ch:ARRAY[1..maxlen]OF char;
        Curlen:0..maxlen
    END;
```

自测题4-4

已知：s='(xyz)+*'，t='(x+z)*y'。试利用联接（||），求子串（substr(s，i，j)）和置换（replace(s_1，i，j，s_2)）等基本操作，将s转化为t。

自测题4-5

已知模式串t='abcaabbabcab'，写出用KMP法求得的每个字符对应的next和nextval函数值。

自测题4-6

试利用下列栈和串的基本操作完成下述填空题。

initstack(s)　　置s为空栈；
push(s，x)　　元素x入栈；

pop(s)　　出栈操作；
gettop(s)　　返回栈顶元素；
sempty(s)　　判栈空函数；
setnull(st)　　置串st为空串；
length(st)　　返回串st的长度；
equal(s1，s2)　　判串s1和s2是否相等的函数；
concat(s1，s2)　　返回联接s1和s2之后的串；
sub(s，i，1)　　返回s中第i个字符；
empty(st)　　判串空函数。

```
FUNC invert(pre:string;VAR exp:string):boolean;
{若给定的表达式的前缀式pre正确，则本过程求得和它相应的表达式exp并返回"true"，否则exp为
空串，并返回"false"。已知原表达式中不包含弧，opset为运算符的集合。}
VAR s:stack;
        i，n:integer;
        succ:boolean;
        ch:char;
BEGIN
        i:=1;n:=length(pre);succ:=true;
             (1)              ;
             (2)              ;
        WHILE(i<n)AND succ DO
        BEGIN
            ch:=sub(pre，i，1);
            IF   (3)    THEN    (4)
            ELSE IF      (5)      THEN    (6)
            ELSE BEGIN
                exp:=concat(   (7)   ,   (8)   );
                exp:=concat(   (9)   ,   (10)   );
                          (11)
            END;
            i:=i+1
        END;
        IF              (12)
        THEN BEGIN
            exp:=concat(exp，sub(ure，n，1));
            invert:=true
        END
        ELSE BEGIN
            SETNULL(exp);
            invert:=false
        END
END;
```

注：每个空格只填一个语句。

4.6 自测题答案

【自测题4-1】

本题可用代入法求解。

答案为D。

【自测题4-2】

非平凡子串的个数为2+3+…+n=n(n+1)/2-1

答案为D。

【自测题4-3】

```
FUNC replace(VAR s1:strtp;i:1..maxlen;j:0..maxlen;s2:strtp)
{  将s1中从位置i开始，长度为j的子串替换为s2。
   替换成功则返回新的长度，否则返回-1}
VAR k: INTEGER;
BEGIN
     IF i+j>s1.curlen THEN RETURN-1;
     IF j>s2.curlen    THEN
          FOR k:=i+j TO s1.curlen DO
               s1.ch[k+s2.curlen-j]:=s1.ch[k];
     ELSE
          FOR k:=s1.curlen DOWNTO i+j DO
               s1.ch[k+s2.curlen-j]:=s1.ch[k];
     FOR k:=1 TO s2..curlen DO
          s1.ch[i+k-1]:=s2.ch[k];
     s1.curlen:=s1.curlen-j+s2.curlen;
END
```

【自测题4-4】

```
s1=substr(s, 1, 5)          {s1='(xyz) '}
s2=substr(s, 3, 1)          {s2='y'}
s3=substr(s, 6, 1)          {s3='+'}
s4=substr(s, 7, 1)          {s4='*'}
replace(s1, 3, 1, s3)       {s1=' (x+z) '}
s=s1 || s4 || s2                {s=' (x+z)*y'}
```

【自测题4-5】

```
next:       0  1  1  1  2  2  3  1  2  3  4  5
nextval:0  1  1  0  2  1  3  0  1  1  0  5
```

【自测题4-6】

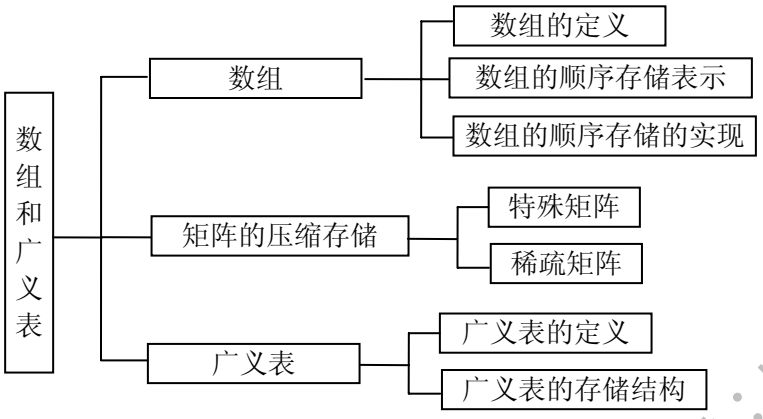
(1) initstack(s)

(2) setnull(exp)

(3) ch in opset
(4) push(s, ch)
(5) sempty(s)
(6) succ:=false
(7) exp
(8) ch
(9) exp
(10) gettop(s)
(11) pop(s)
(12) sempty(s)

第 5 章　数组和广义表

5.1　基本知识结构图



5.2　知 识 点

1. 数组的存储结构

一旦建立了数组，则结构中的数据元素个数和元素之间的关系就不再发生变动，而且数组一般不做插入或删除操作，因此，常采用顺序存储结构。

2. 特殊矩阵的压缩存储

压缩存储是指为多个值相同的元分配一个存储空间，对零元不分配空间。

在特殊矩阵中，因非零元的分布有一个明显的规律，可将非零元存储到一个一维数组中，并找到每个非零元在一维数组中的对应关系。

3. 稀疏矩阵的压缩存储

通常认为，在一个矩阵中非零元素的个数与总元素的个数比小于5%，称该矩阵为稀疏矩阵。可用三元组表和十字链表的方法存储稀疏矩阵。

4. 广义表及其存储结构

广义表一般记作：$LS=(d_1, d_2, \cdots, d_n)$，其中，$LS$是广义表$(d_1, d_2, \cdots, d_n)$的名称；$n$

是它的长度；d_i可以是单个元素，也可以是广义表，分别称为广义表LS的原子和子表。广义表的定义是一个递归定义。当广义表非空时，称第一个元素d_1为LS的表头（Head），称其余元素组成的表$(d_2, d_3, \cdots, d_n)$是LS的表尾（Tail）。

由于广义表中的数据元素可以具有不同的结构（原子或列表），因此难以用顺序存储结构表示，通常采用链式存储结构，每个数据元素可用一个结点表示。结点的结构有两种：表结点（用于表示列表）和原子结点（用于表示原子）。

5.3　习题及参考答案

1．假设有二维数组$A_{6\times8}$，每个元素用相邻的6个字节存储，存储器按字节编址。已知A的起始存储位置（基地址）为1000，计算：

（1）数组A的体积（即存储量）；

（2）数组A的最后一个元素a_{57}的第一个字节的地址；

（3）按行存储时，元素a_{14}的第一个字节的地址；

（4）按列存储时，元素a_{47}的第一个字节的地址。

【解答】（1）288字节；（2）1282；（3）1072；（4）1276。

2．假设按低下标优先存储整数数组$A_{9\times3\times5\times8}$时，第一个元素的字节地址是100，每个整数占四个字节。问下列元素的存储地址是什么？

（1）a_{0000}（2）a_{1111}（3）a_{3125}（4）a_{8247}

【解答】（1）100；（2）676；（3）1784；（4）4612。

3．按高下标优先存储方式（以最右的下标为主序），顺序列出数组$A_{2\times2\times3\times3}$中所有元素$a_{ijkl}$，为了简化表达，可以只列出（$i$，$j$，$k$，$l$）的序列。

【解答】（0，0，0，0），（1，0，0，0），（0，1，0，0），（1，1，0，0），…，（0，1，2，2），（1，1，2，2）。

4．设有上三角矩阵$(a_{ij})_{n\times n}$，将其上三角元素逐行存于数组$B[m]$中（m充分大），使得$B[k]=a_{ij}$且$k=f_1(i)+f_2(j)+c$。试推导出函数f_1，f_2和常数c（要求f_1和f_2中不含常数项）。

【解答】$k=ni-(n-j)-\dfrac{i(i-1)}{2}-1$，（$i\leqslant j$）

则得$f_1(i)=(n+\dfrac{1}{2})i-\dfrac{1}{2}i^2$，$f_2(j)=j$，$c=-(n+1)$。

5．设有三对角矩阵$(a_{ij})_{n\times n}$，将其三条对角线上的元素存于数组$B[3][n]$中，使得元素$B[u][v]=a_{ij}$，试推导出从（i，j）到（u，v）的下标变换公式。

【解答】一种答案是：$u=i-j+1$，$v=j-1$。

6．设有三对角矩阵$(a_{ij})_{n\times n}$，将其三条对角线上的元素逐行地存于数组$B[3n-2]$中，使得$B[k]=a_{ij}$，求：

（1）用i，j表示k的下标变换公式；

（2）用k表示i，j的下标变换公式。

【解答】（1）$k=2(i-1)+j-1$，$(|\ i-j\ |\leqslant 1)$

（2）$i=(k+1)$ DIV $3+1$，$(0\leqslant k\leqslant 3n-1)$

$j=k+1-2(i-1)\ =\ k+1-2(k$ DIV $3)$

7．假设一个准对角矩阵

$$\begin{bmatrix} a_{11} & a_{12} & & & & & & & \\ a_{21} & a_{22} & & & & & & & \\ & & a_{33} & a_{34} & & & & & \\ & & a_{43} & a_{44} & & & & & \\ & & & & \cdots & & & & \\ & & & & & a_{i,\ j} & & & \\ & & & & & & \cdots & & \\ & & & & & & & a_{2m-1,\ 2m-1} & a_{2m-1,\ 2m} \\ & & & & & & & a_{2m,\ 2m-1} & a_{2m,\ 2m} \end{bmatrix}$$

按以下方式存于一维数组$B[4m]$中：

0	1	2	3	4	5	6		k		$4m-2$	$4m-1$
a_{11}	a_{12}	a_{21}	a_{22}	a_{33}	a_{34}	a_{43}	…	$a_{i,j}$	…	$a_{2m-1,\ 2m}$	$a_{2m,\ 2m-1}$	$a_{2m,\ 2m}$

写出由一对下标（i，j）求k的转换公式。

【分析】考点为计算数组某元素的存储地址，注意事项有4点：①已知元素的地址；②按行还是按列存储；③每个单元所占的字节数；④所要求元素与已知元素的相对位置关系。

【解答】i为奇数时 $k=i+j-2$

i为偶数时 $k=i+j-1$

合并成一个公式，可写成

$k=i+j-(i\%2)-1$

或 $k=2(i$ DIV $2)+j-1$

8．已知A为稀疏矩阵，试从空间和时间角度比较采用两种不同的存储结构（二维数组和三元组表）完成求$\sum_{i=1}^{n}a_{ii}$运算的优缺点。

【解答】利用二维数组来存储稀疏矩阵，因为要存储大量的零元素，所以空间利用率低，而用三元组表可以节省大量的存储空间，但是时间复杂度提高了。

9．求下列广义表操作的结果：

（1）GetHead【(*p*，*h*，*w*)】；
（2）GetTail【(*b*，*k*，*p*，*h*)】；
（3）GetHead【((*a*，*b*)，(*c*，*d*))】；
（4）GetTail【((*a*，*b*)，(*c*，*d*))】；
（5）GetHead【GetTail【((*a*，*b*)，(*c*，*d*))】】；
（6）GetTail【GetHead【((*a*，*b*)，(*c*，*d*))】】；
（7）GetHead【GetTail【GetHead【((*a*，*b*)，(*c*，*d*))】】】；
（8）GetTail【GetHead【GetTail【((*a*，*b*)，(*c*，*d*))】】】。

注：【】是函数的符号。

【解答】（1）*p*；　　（2）(*k*，*p*，*h*)；　　（3）(*a*，*b*)；　　（4）((*c*，*d*))；
（5）(*c*，*d*)；　　（6）(*b*)；　　（7）*b*；　　（8）(*d*)。

10．利用广义表的GetHead和GetTail操作写出如上题的函数表达式，把原子*banana*分别从下列广义表中分离出来。

（1）L_1=(*apple*，*pear*，*banana*，*orange*)；
（2）L_2=((*apple*，*pear*)，(*banana*，*orange*))；
（3）L_3=(((*apple*)，(*pear*)，(*banana*)，(*orange*)))；
（4）L_4=(*apple*，(*pear*)，((*banana*))，(((*orange*))))；
（5）L_5=((((*apple*)))，((*pear*))，(*banana*)，*orange*)；
（6）L_6=((((*apple*)，*pear*)，*banana*)，*orange*)；
（7）L_7=(*apple*，(*pear*，(*banana*)，*orange*))

【解答】（1）GetHead【GetTail【GetTail【L_1】】】；
（2）GetHead【GetHead【GetTail【L_2】】】；
（3）GetHead【GetHead【GetTail【GetTail【GetHead【L_3】】】】】；
（4）GetHead【GetHead【GetHead【GetTail【GetTail【L_4】】】】】；
（5）GetHead【GetHead【GetTail【GetTail【L_5】】】】；
（6）GetHead【GetTail【GetHead【L_6】】】；
（7）GetHead【GetHead【GetTail【GetHead【GetTail【L_7】】】】】。

11．已知以下各图为广义表的存储结构图，其中表结点结构为

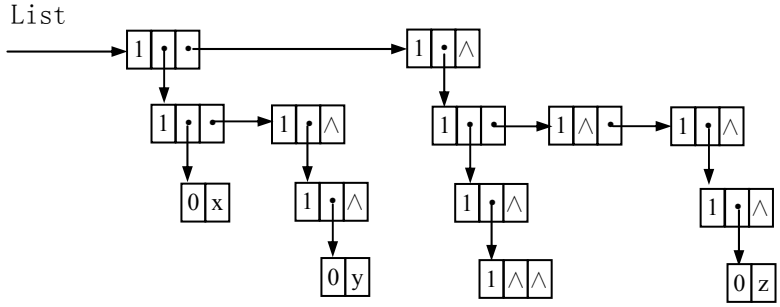
tag=1	hp	tp

，原子结点结构为

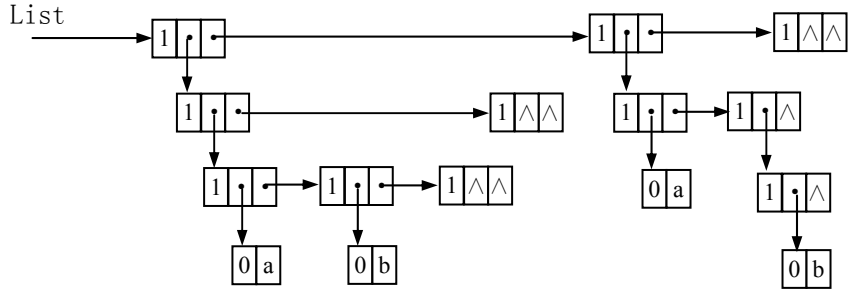
tag=0	atom

。写出各图表示的广义表。

(1)



(2)



【解答】（1）((*x*, (*y*)), ((((), (), (*z*)))

（2）(((*a*, *b*, ()), ()), (*a*, (*b*)), ())

12．已知等差数列的第一项为a_1，公差为*d*，试写出该数列前*n*项的和*S*(*n*)（*n*≥0）的递归定义。

【解答】$S(n)=S(n-1)+a_n=S(n-1)+(n-1)d+a_1$

13．写出求给定集合的幂集的递归定义。

【解答】用*P*(*A*)表示*A*的幂集，定义谓词beset(*X*)表示*X*是一个集合，则

$$P(A)=\begin{cases}\{\Phi\} & \text{当}|A|=0\text{时}\\ P(A-\{a\})\cup I(P(A-\{a\}),\ a) & \text{当}|A|\neq 0\text{时}\end{cases}$$

其中，$a\in A$且$I(X,\ y)=\{x\mid(\exists\ x_0)(x_0\in X\wedge \text{beset}(x_0)\wedge x=x_0\cup\{y\})\}$。

14．试利用C语言中的增量运算“++”和减量运算“--”写出两个非负整数*a*和*b*相加的递归定义。

【解答】记*a*+*b*为add(*a*, *b*)，一种写法为

$$\text{add}(a,\ b)=\begin{cases}a & \text{当}b=0\text{时}\\ \text{add}(++a,\ --b) & \text{当}b>0\text{时}\end{cases}$$

15．试设计一个算法，将数组A_n中的元素*A*[0]至*A*[*n*-1]循环右移*k*位，并要求只用一个元素大小的附加存储，元素移动或交换次数为*O*(*n*)。

【分析】本题难点在于找到*O*(*n*)的算法。注意分析研究此问题的数学性质，以寻求好

的算法。本题有多种解法，但要注意不要将在特殊条件下出现的现象当作一般规律。例如，本题易犯的错误是，从第i个分量中的元素起，循环右移k位，顶掉第$(j+k)\%n$个元素，依次类推。这似乎是个漂亮的算法，但只是在某些情况下可得正确结果。

【解答】算法如下：

```
void RSh(int A[n], int k)//把数组A的元素循环右移k位，只用一个辅助存储空间
{
  for(i=1;i<=k;i++)
    if(n%i==0&&k%i==0) p=i;//求n和k的最大公约数p
  for(i=0;i<p;i++)
  {
    j=i;l=(i+k)%n;temp=A[i];
    while(l!=i)
    {
      A[j]=temp;
      temp=A[l];
      A[l]=A[j];
      j=l;l=(j+k)%n;
    }// 循环右移一位
    A[i]=temp;
  }//for
}//RSh
```

16．若矩阵$A_{m\times n}$中的某个元素a_{ij}是第i行中的最小值，同时又是第j列中的最大值，则称此元素为该矩阵中的一个马鞍点。假设以二维数组存储矩阵$A_{m\times n}$，试编写求出矩阵中所有马鞍点的算法，并分析你的算法在最坏情况下的时间复杂度。

【分析】可尝试多种解法，例如可设置两个辅助数组max[n]、min[m]，通过二重循环找到每一列的最大值和每一行的最小值，记录在max和min中，再扫描max、min数组，找到马鞍点。注意矩阵中若存在多个马鞍点，则必定相等，但反之，值相等的点不一定都是马鞍点。本题可有多种解法，最坏情况下的时间复杂度可达$O(m\times n)$。

【解答】算法如下：

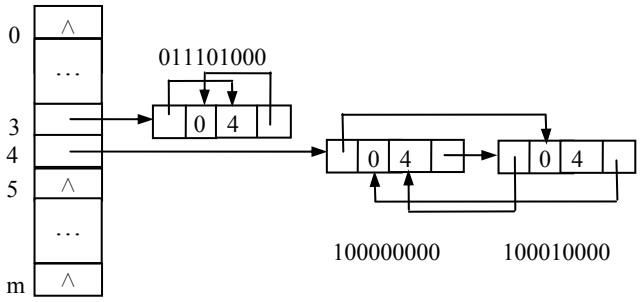
```
void Get_Saddle(int A[m][n])//求矩阵A中的马鞍点
{
  for(i=0;i<m;i++)
  {
    for(min=A[i][0], j=0;j<n;j++)
      if(A[i][j]<min) min=A[i][j]; //求一行中的最小值
    for(j=0;j<n;j++)
      if(A[i][j]==min) //判断这个（些）最小值是否马鞍点
      {
        for(flag=1, k=0;k<m;k++)
          if(min<A[k][j]) flag=0;
             if(flag) printf("Found a saddle element!\nA[%d][%d]=%d", i, j, A[i][j]);
      }
  }//for
}//Get_Saddle
```

17．假设稀疏矩阵*A*和*B*均以三元组顺序表作为存储结构。试写出矩阵相加的算法，另设三元组表*C*存放结果矩阵。

【解答】算法如下：

```
void TSMatrix_Add(TSMatrix A, TSMatrix B, TSMatrix &C)//三元组表示的稀疏矩阵加法
{
  C.mu=A.mu;C.nu=A.nu;C.tu=0;
  pa=1;pb=1;pc=1;
  for(x=1;x<=A.mu;x++) //对矩阵的每一行进行加法
  {
    while(A.data[pa].i<x) pa++;
    while(B.data[pb].i<x) pb++;
    while(A.data[pa].i==x&&B.data[pb].i==x)//行列值都相等的元素
    {
      if(A.data[pa].j==B.data[pb].j)
      {
        ce=A.data[pa].e+B.data[pb].e;
        if(ce) //和不为0
        {
          C.data[pc].i=x;
          C.data[pc].j=A.data[pa].j;
          C.data[pc].e=ce;
          pa++;pb++;pc++;
        }
      }//if
      else if(A.data[pa].j>B.data[pb].j)
         {
             C.data[pc].i=x;
             C.data[pc].j=B.data[pb].j;
             C.data[pc].e=B.data[pb].e;
             pb++;pc++;
         }
         else
         {
             C.data[pc].i=x;
             C.data[pc].j=A.data[pa].j;
             C.data[pc].e=A.data[pa].e;
             pa++;pc++;
         }
    }//while
    while(A.data[pa]==x) //插入A中剩余的元素(第x行)
    {
      C.data[pc].i=x;
      C.data[pc].j=A.data[pa].j;
      C.data[pc].e=A.data[pa].e;
      pa++;pc++;
    }
    while(B.data[pb]==x) //插入B中剩余的元素(第x行)
    {
```

```
        C.data[pc].i=x;
        C.data[pc].j=B.data[pb].j;
        C.data[pc].e=B.data[pb].e;
        pb++;pc++;
      }
    }//for
    C.tu=pc;
}//TSMatrix_Add
```



结果矩阵

18．假设系数矩阵*A*和*B*均为以三元组顺序表作为存储结构。试写出满足以下条件的矩阵相加的算法：假设三元组顺序表*A*的空间足够大，将矩阵*B*加到矩阵*A*上，不增加*A*，*B*之外的附加空间，你的算法能否达到$O(m+n)$的时间复杂度？其中*m*和*n*分别为*A*，*B*矩阵中非零元的数目。

【分析】先将A.data[1..A.tu]搬到A.data[MAXSIZE-A.tu+1..MAXSIZE]再进行处理，可以得到$O(m+n)$算法。当然也可以从后向前处理，最后上移。

【解答】算法如下：

```
void TSMatrix_Addto(TSMatrix &A, TSMatrix B)//将三元组矩阵B加到A上
{
  for(i=1;i<=A.tu;i++)
    A.data[MAXSIZE-A.tu+i]=A.data[i];//把A的所有元素都移到尾部以腾出位置
  pa=MAXSIZE-A.tu+1;pb=1;pc=1;
  for(x=1;x<=A.mu;x++) //对矩阵的每一行进行加法
  {
    while(A.data[pa].i<x) pa++;
    while(B.data[pb].i<x) pb++;
    while(A.data[pa].i==x&&B.data[pb].i==x)//行列值都相等的元素
    {
      if(A.data[pa].j==B.data[pb].j)
      {
        ne=A.data[pa].e+B.data[pb].e;
        if(ne) //和不为0
        {
          A.data[pc].i=x;
          A.data[pc].j=A.data[pa].j;
          A.data[pc].e=ne;
          pa++;pb++;pc++;
```

```
        }
      }//if
      else if(A.data[pa].j>B.data[pb].j)
         else if(A.data[pa].j>B.data[pb].j)
           {
                  A.data[pc].i=x;
                  A.data[pc].j=B.data[pb].j;
                  A.data[pc].e=B.data[pb].e;
                  pb++;pc++;
           }
           else
           {
                 A.data[pc].i=x;
                 A.data[pc].j=A.data[pa].j;
                 A.data[pc].e=A.data[pa].e;
                 pa++;pc++;
           }
    }//while
    while(A.data[pa]==x) //插入A中剩余的元素(第x行)
    {
      A.data[pc].i=x;
      A.data[pc].j=A.data[pa].j;
      A.data[pc].e=A.data[pa].e;
      pa++;pc++;
    }
    while(B.data[pb]==x) //插入B中剩余的元素(第x行)
    {
      A.data[pc].i=x;
      A.data[pc].j=B.data[pb].j;
      A.data[pc].e=B.data[pb].e;
      pb++;pc++;
    }
  }//for
  A.tu=pc;
  for(i=A.tu;i<MAXSIZE;i++) A.data[i]={0, 0, 0}; //清除原来的A中记录
}//TSMatrix_Addto
```


19．三元组顺序表中的一种变型是，从三元顺序表中去掉行下标域得到二元组顺序表，另设一个行起始向量，其每个分量是二元组顺序表的一个下标值，指示该行中第一个非零元素在二元组顺序表中的起始位置。试编写一个算法，由矩阵元素的下标值i，j求矩阵元素。试讨论这种方法和三元组顺序表相比有什么优缺点。

【分析】这种表示方法的优点是，可以随机存取稀疏矩阵中任意一行的非零元，而三元组顺序表只能按行进行顺序存取。

【解答】算法如下：

```
typedef struct{
                    int j;
                    int e;
                  } DSElem;
```

```
typedef struct{
                DSElem data[MAXSIZE];
                int cpot[MAXROW];//这个向量存储每一行在二元组中的起始位置
                int mu, nu, tu;
              } DSMatrix; //二元组矩阵类型
Status DSMatrix_Locate(DSMatrix A, int i, int j, int &e)//求二元组矩阵的元素A[i][j]的值
e
{
  for(s=cpot[i];s<cpot[i+1]&&A.data[s].j!=j;s++);//注意查找范围
  if(A.data[s].i==i&&A.data[s].j==j) //找到了元素A[i][j]
  {
    e=A.data[s];
    return OK;
  }
  return ERROR;
}//DSMatrix_Locate
```

20．三元组顺序表的另一种变型是，不存矩阵元素的行、列下标，而存非零元在矩阵中以行为主序时排列的顺序号，即在LOC(0，0)=1，I=1时按公式 $\mathrm{LOC}(j_1,j_2,\Lambda\ ,j_n)=\mathrm{LOC}(0,0,\Lambda\ ,0)+\sum_{i=1}^{n}c_i j_i$ 计算出的值。试写一算法，由矩阵元素的下标值i，j求元素的值。

【解答】算法如下：

```
typedef struct{
                int seq; //该元素在以行为主序排列时的序号
                int e;
           } SElem;
typedef struct{
                SElem data[MAXSIZE];
                int mu, nu, tu;
            } SMatrix; //单下标二元组矩阵类型
Status SMatrix_Locate(SMatrix A, int i, int j, int &e)//求单下标二元组矩阵的元素A[i][j]
的值e
{
    s=i*A.nu+j+1;p=1;
    while(A.data[p].seq<s) p++; //利用各元素seq值逐渐递增的特点
    if(A.data[p].seq==s) //找到了元素A[i][j]
    {
        e=A.data[p].e;
        return OK;
    }
    return ERROR;
}//SMatrix_Locate
```

21．若将稀疏矩阵A的非零元素以行序为主序的顺序存于一维数组V中，并用二维数组B表示A中的相应元素是否为零元素（以0和1分别表示零元素和非零元素）。

例如，

$A=\begin{bmatrix}15 & 0 & 0 & 22\\ 0 & -6 & 0 & 0\\ 91 & 0 & 0 & 0\end{bmatrix}$可用$I$=(15，22，-6，9)和$B=\begin{bmatrix}1 & 0 & 0 & 1\\ 0 & 1 & 0 & 0\\ 1 & 0 & 0 & 0\end{bmatrix}$表示。

试写一算法，在上述表示法中实现矩阵相加的运算。并分析你的算法的时间复杂度。

【分析】注意矩阵B的作用，即构造两个B矩阵相与的结果，全为1时才进行I中元素运算。

【解答】算法如下：

```
typedef enum{0, 1} bool;
typedef struct{
                    int mu, nu;
                    int elem[MAXSIZE];
                    bool map[mu][nu];
                  } BMMatrix; //用位图表示的矩阵类型
void BMMatrix_Add(BMMatrix A, BMMatrix B, BMMatrix &C)//位图矩阵的加法
{
  C.mu=A.mu;C.nu=A.nu;
  pa=1;pb=1;pc=1;
  for(i=0;i<A.mu;i++) //每一行的相加
    for(j=0;j<A.nu;j++) //每一个元素的相加
    {
      if(A.map[i][j]&&B.map[i][j]&&(A.elem[pa]+B.elem[pb]))//结果不为0
      {
        C.elem[pc]=A.elem[pa]+B.elem[pb];
        C.map[i][j]=1;
        pa++;pb++;pc++;
      }
      else if(A.map[i][j]&&!B.map[i][j])
          {
              C.elem[pc]=A.elem[pa];
              C.map[i][j]=1;
              pa++;pc++;
          }
          else if(!A.map[i][j]&&B.map[i][j])
              {
                  C.elem[pc]=B.elem[pb];
                  C.map[i][j]=1;
                  pb++;pc++;
              }
    }
}//BMMatrix_Add
```

22．试编写一个以三元组形式输出用十字链表表示的稀疏矩阵中非零元素及其下标的算法。

【解答】算法如下：

```
void Print_OLMatrix(OLMatrix A)//以三元组格式输出十字链表表示的矩阵
{
  for(i=0;i<A.mu;i++)
  {
    if(A.rhead[i])
      for(p=A.rhead[i];p;p=p->right) //逐次遍历每一个行链表
        printf("%d %d %d\n", i, p->j, p->e);
  }
}//Print_OLMatrix
```

23. 试按十字链表存储表示编写将稀疏矩阵*B*加到稀疏矩阵*A*上的算法。

【解答】算法如下：

```
void OLMatrix_Add(OLMatrix &A, OLMatrix B)//把十字链表表示的矩阵B加到A上
{
  for(j=1;j<=A.nu;j++) cp[j]=A.chead[j]; //向量cp存储每一列当前最后一个元素的指针
  for(i=1;i<=A.mu;i++)
  {
    pa=A.rhead[i];pb=B.rhead[i];pre=NULL;
    while(pb)
    {
      if(pa==NULL || pa->j>pb->j) //新插入一个结点
      {
        p=(OLNode*)malloc(sizeof(OLNode));
        if(!pre) A.rhead[i]=p;
        else pre->right=p;
        p->right=pa;pre=p;
        p->i=i;p->j=pb->j;p->e=pb->e; //插入行链表中
        if(!A.chead[p->j])
        {
          A.chead[p->j]=p;
          p->down=NULL;
        }
        else
        {
          while(cp[p->j]->down) cp[p->j]=cp[p->j]->down;
          p->down=cp[p->j]->down;
          cp[p->j]->down=p;
        }
        cp[p->j]=p; //插入列链表中
      }//if
      else if(pa->j<pb->j)
          {
             pre=pa;
             pa=pa->right;
          } //pa右移一步
          else if(pa->e+pb->e)
              {
                   pa->e+=pb->e;
                   pre=pa;pa=pa->right;
```

```
                pb=pb->right;
            } //直接相加
            else
            {
                if(!pre) A.rhead[i]=pa->right;
                else pre->right=pa->right;
                p=pa;pa=pa->right; //从行链表中删除
                if(A.chead[p->j]==p)
                    A.chead[p->j]=cp[p->j]=p->down;
                else cp[p->j]->down=p->down; //从列链表中删除
                free (p);
             }//else
    }//while
  }//for
}//OLMatrix_Add
```

24. 采用*m*元多项式的表示方式，写一个求*m*元多项式中第一变元的偏导数的算法。

【解答】算法如下：

```
void MPList_PianDao(MPList &L)//对广义表存储结构的多元多项式求第一变元的偏导数的算法。
{
  for(p=L->hp->tp;p&&p->exp;pre=p, p=p->tp)
  {
    if(p->tag) Mul(p->hp, p->exp);
    else p->coef*=p->exp; //把指数乘在本结点或其下属结点上
    p->exp--;
  }
  pre->tp=NULL;
  if(p) free (p); //删除可能存在的常数项
}//MPList_PianDao
void Mul(MPList &L, int x)//递归算法，对多元多项式L乘以x
{
  for(p=L;p;p=p->tp)
  {
    if(!p->tag) p->coef*=x;
    else Mul(p->hp, x);
  }
}//Mul
```

25. 采用*m*元多项式的表示方式，写一个*m*元多项式相加的算法。

【解答】算法如下：

```
void MPList_Add(MPList A, MPList B, MPList &C)//广义表存储结构的多项式相加的递归算法
{
  C=(MPLNode*)malloc(sizeof(MPLNode));
  if(!A->tag&&!B->tag) //原子项，可直接相加
  {
    C->coef=A->coef+B->coef;
    if(!C->coef)
    {
```

```
    free(C);
    C=NULL;
  }
}//if
else if(A->tag&&B->tag) //两个多项式相加
   {
      p=A;q=B;pre=NULL;
      while(p&&q)
      {
         if(p->exp==q->exp)
         {
            C=(MPLNode*)malloc(sizeof(MPLNode));
            C->exp=p->exp;
            MPList_Add(A->hp, B->hp, C->hp);
            pre->tp=C;pre=C;
            p=p->tp;q=q->tp;
         }
        else if(p->exp>q->exp)
           {
               C=(MPLNode*)malloc(sizeof(MPLNode));
               C->exp=p->exp;
               C->hp=A->hp;
               pre->tp=C;pre=C;
               p=p->tp;
           }
           else
           {
               C=(MPLNode*)malloc(sizeof(MPLNode));
               C->exp=q->exp;
               C->hp=B->hp;
               pre->tp=C;pre=C;
               q=q->tp;
           }
      }//while
      while(p)
      {
           C=(MPLNode*)malloc(sizeof(MPLNode));
           C->exp=p->exp;
           C->hp=p->hp;
           pre->tp=C;pre=C;
           p=p->tp;
      }
      while(q)
      {
           C=(MPLNode*)malloc(sizeof(MPLNode));
           C->exp=q->exp;
           C->hp=q->hp;
           pre->tp=C;pre=C;
           q=q->tp;
      } //将其同次项分别相加得到新的多项式
```

```
  }//else if
  else if(A->tag&&!B->tag) //多项式和常数项相加
  {
    x=B->coef;
    for(p=A;p->tp->tp;p=p->tp);
    if(p->tp->exp==0) p->tp->coef+=x; //当多项式中含有常数项时，加上常数项
    if(!p->tp->coef)
    {
      free(p->tp);
      p->tp=NULL;
    }
    else
    {
      q=(MPLNode*)malloc(sizeof(MPLNode));
      q->coef=x;q->exp=0;
      q->tag=0;q->tp=NULL;
      p->tp=q;
    } //否则新建常数项，下同
     }//else if
     else
     {
          x=A->coef;
          for(p=B;p->tp->tp;p=p->tp);
              if(p->tp->exp==0) p->tp->coef+=x;
              if(!p->tp->coef)
              {
                 free(p->tp);
                 p->tp=NULL;
              }
              else
              {
                 q=(MPLNode*)malloc(sizeof(MPLNode));
                 q->coef=x;q->exp=0;
                 q->tag=0;q->tp=NULL;
                 p->tp=q;
               }
     }//else
}//MPList_Add
```

26．试按表头、表尾的分析方法重写求广义表的深度的递归算法。

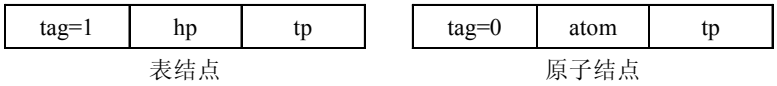
【分析】广义表的深度*DEPTH*(*LS*)也可如下定义：

$$\mathrm{DEPTH(LS)}=\begin{cases}1 & \text{当LS为空表}\\ 0 & \text{当LS为单原子}\\ \mathrm{MAX}\{\mathrm{DEPTH(GetHead(LS))}+1,\ \mathrm{DEPTH(GetTail(LS))}\} & \text{其他情况}\end{cases}$$

【解答】算法如下：

```
int GList_Getdepth(GList L)//求广义表深度的递归算法
{
  if(!L->tag) return 0; //原子深度为0
  else if(!L) return 1; //空表深度为1
  m=GList_Getdepth(L->ptr.hp)+1;
  n=GList_Getdepth(L->ptr.tp);
  return m>n?m:n;
}//GList_Getdepth
```

27. 已知列表的一种结点结构表示如下：



表结点　　原子结点

编写复制广义表的递归算法。

【分析】按题中对结点结构的定义，原子结点也有表尾域，因此不管ls->tag是否为零，都要执行第二个递归调用语句。原子结点是否有表尾域在复制操作意义下的差别是非本质的。

【解答】算法如下：

```
void GList_Copy(GList A, GList &B)//复制广义表的递归算法
{
    if(!A->tag) //当结点为原子时，直接复制
    {
        B->tag=0;
        B->atom=A->atom;
    }
    else //当结点为子表时
    {
        B->tag=1;
        if(A->ptr.hp)
        {
            B->ptr.hp=malloc(sizeof(GLNode));
            GList_Copy(A->ptr.hp, B->ptr.hp);
        } //复制表头
        if(A->ptr.tp)
        {
             B->ptr.tp=malloc(sizeof(GLNode));
             GList_Copy(A->ptr.tp, B->ptr.tp);
        } //复制表尾
    }//else
}//GList_Copy
```

28. 试编写判别两个广义表是否相等的递归算法。

【分析】可以在参数表中加一个值参，以表示当前递归的层次。

【解答】算法如下：

```
int GList_Equal(GList A, GList B)//判断广义表A和B是否相等，是则返回1，否则返回0
{ //广义表相等可分三种情况:
  if(!A&&!B) return 1; //空表是相等的
  if(!A->tag&&!B->tag&&A->atom==B->atom) return 1;//原子的值相等
  if(A->tag&&B->tag)
    if(GList_Equal(A->ptr.hp, B->ptr.hp)&&GList_Equal(A->ptr.tp, B->ptr.tp))
      return 1; //表头表尾都相等
    return 0;
}//GList_Equal
```

29. 试编写递归算法，输出广义表中所有原子项及其所在层次。

【解答】算法如下：

```
void GList_PrintElem(GList A, int layer)//递归输出广义表的原子及其所在层次，layer表示当
前层次
{
  if(!A) return;
  if(!A->tag) printf("%d %d\n", A->atom, layer);
  else
  {
    GList_PrintElem(A->ptr.hp, layer+1);
    GList_PrintElem(A->ptr.tp, layer); //注意尾表与原表是同一层次
  }
}//GList_PrintElem
```

30. 试编写递归算法，逆转广义表中的数据元素。

例如，将广义表(*a*, ((*b*, *c*), ( )), (((*d*), *e*), *f*))

逆转为((*f*, (*e*, (*d*))), (( ), (*c*, *b*)), *a*)。

【分析】一个简单的分析方法是，将广义表看成是一个“线性链表”，则其逆转的过程和线性链表的逆转类似。

【解答】算法如下：

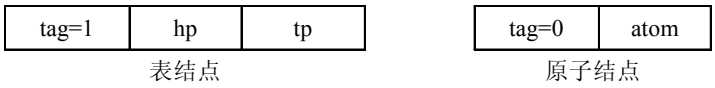
```
void GList_Reverse(GList A)//递归逆转广义表A
{
    if(A->tag&&A->ptr.tp) //当A不为原子且表尾非空时才需逆转
    {
        D=C->ptr.hp;
        A->ptr.tp->ptr.hp=A->ptr.hp;A->ptr.hp=D; //交换表头和表尾
        GList_Reverse(A->ptr.hp);
        GList_Reverse(A->ptr.tp); //逆转表头和表尾
    }
}//GList_Reverse
```

31．假设广义表按如下形式的字符串表示。

$(a_1, a_2, \cdots, a_n)$　$n \geqslant 0$

其中，a_i或为单字母表示的原子，或为广义表；n=0时为只含空格字符的空表()。

列表的链表结点结构如下：



按照读入的一个广义表字符串建立其存储结构的递归算法。

【分析】若广义表为空表，则其输入形式为'()'，其特点是在左括弧之后输入的是空格字符''；若广义表非空，则在左括弧之后输入的必为表头，它或是表示原子的单字母，或是子表的首字符'('，其他均为非法字符。对非空的广义表，在表头之后输入的合法字符或为')'，或为','，前者表明表尾为空表，后者表明表尾为非空表。由此分析可得下列概要算法。

【解答】算法如下：

```
Status CtrLtBU(GList &Ls){
    //本算法识别一个待输入的广义表字符串，单字母表示原子，按题中所示结构形式建立相应存
    //储结构。调用本算法之前应先执行scanf (ch)语句，以滤去输入串中的字符'（'或'，'，算
法执行
    //结束时，ch中含所识别的广义表字符串中最后的字符'）'。
  scanf (ch);
    if (ch = ='')
    {//建空表;
      scanf (ch);
      if (ch!= '）') return ERROR;
      else return OK;
    }
    if (!(Ls= (GList ) malloc (sizeof (GLNode)))) return OVERFLOW;
    else Ls->tag=1;    //建非空广义表
    if (ch是字母)   { 建原子结点  *ls->ptr.hp;}
    else if (ch = =' (')   CrtLtBU(Ls->ptr.hp);   //建子表
        else return ERROR;
    scanf (ch);          //输入表头之后的字符
    if (ch = = '）') ls->ptr.tp=NULL;     //表尾为空表
    else if (ch = = ', ') CrtLtBU(ls->ptr.tp);    //建表尾
        else return ERROR;
    return OK;
}  //CrtLtBU
```

32．试编写递归算法，删除广义表中所有值等于x的原子项。

【分析】注意删除原子项并不仅仅是删除该原子结点。

【解答】算法如下：

```
void GList_DelElem(GList &A, int x)//从广义表A中删除所有值为x的原子
{
  if(A->ptr.hp->tag) GList_DelElem(A->ptr.hp, x);
  else if(!A->ptr.hp->tag&&A->ptr.hp->atom==x)
       {
          q=A;
          A=A->ptr.tp;    //删去元素值为x的表头
          free(q);
          GList_DelElem(A, x);
       }
  if(A->ptr.tp) GList_DelElem(A->ptr.tp, x);
}//GList_DelElem
```

33. 试编写算法，依次从左至右输出广义表中第*l*层的原子项。

例如，广义表(*a*, (*b*, (*c*)), (*d*))中的*a*为第一层的原子项；*b*和*d*为第二层的原子项；*c*为第三层的原子项。

【分析】层序遍历的问题一般都是借助队列来完成的，每次从队头取出一个元素的同时把它下一层的孩子插入队尾，这是层序遍历的基本思想。

【解答】算法如下：

```
void GList_PrintElem_LOrder(GList A)//按层序输出广义表A中的所有元素
{
  InitQueue(Q);
  for(p=L;p;p=p->ptr.tp) EnQueue(Q, p);
  while(!QueueEmpty(Q))
  {
    DeQueue(Q, r);
    if(!r->tag) printf("%d", r->atom);
    else
      for(r=r->ptr.hp;r;r=r->ptr.tp) EnQueue(Q, r);
  }//while
}//GList_PrintElem_LOrder
```


5.4 考研真题分析

例题5-1 （中国科学技术大学1998年试题）

在一维数组*A*中存放着*m*+*n*个整数（*m*，*n*≥1），如图5-1所示的流程图给出了一种算法，使*A*中前*m*个元素（*A*[0]-*A*[*m*-1]），与后*n*个元素（*A*[*m*]-*A*[*m*+*n*]）互换位置，并保持其各自原有的内部顺序。例如，若*A*如表5-1所示，则互换后的内容变为如表5-2所示。试填写流程图中的空格1~5，使之变为完善的算法框图。

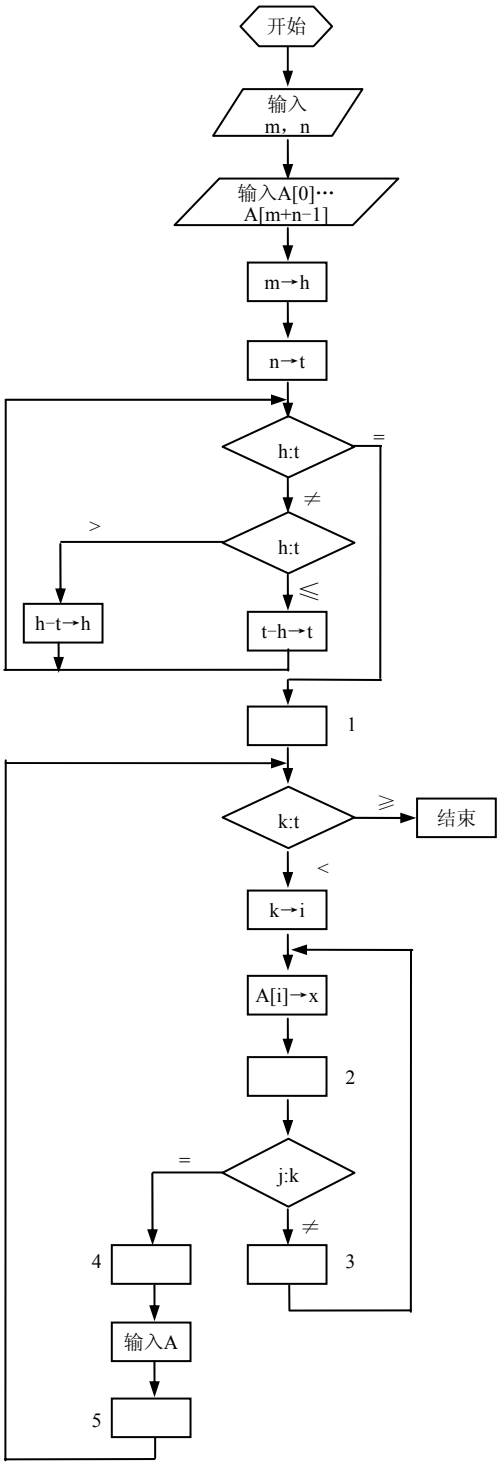


图 5-1　流程图

表5-1　互换前的数组A

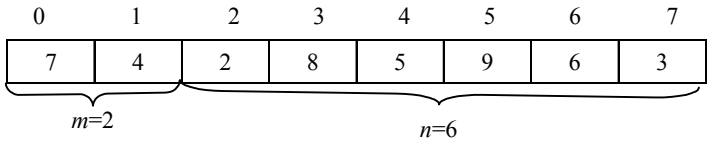
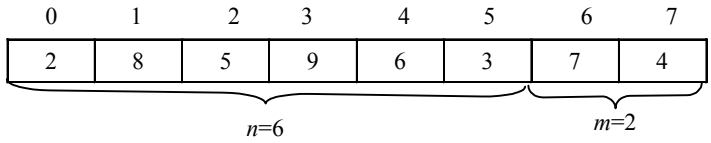


表5-2　互换后的数组A



【解答】

第1空：0→k；　　第2空：(i+m)MOD(m+n) →j；

第3空：j→i；　　第4空：x→A[i]；　　第5空：k+1→k

例题5-2　（中国科学院软件研究所1998年试题）

若广义表*A*满足Head(A)=Tail(A)，则*A*为_____。

A. ()　　B. (())　　C. (()，())　　D. (()，())，())

【解答】B

例题5-3　（中国科学技术大学1998年试题）

设广义表*L*=((), ())，则Head(L)=_____；Tail(L)=_____；*L*的长度是____；*L*的深度是_____。

【分析】

对于广义表$LS=(a_1, a_2, a_3, \cdots, a_n)$，各个运算如下。

表头：　Head(LS)=a_1

表尾：　Tail(LS)=($a_2, a_3, \cdots, a_n$)

长度：　Length(LS)=n

深度：　Depth(LS)=1　　当LS为空表

Depth(LS)=0　　当LS为原子

Depth(LS)=1+MAX{Depth(a_i)|1, 2, …, n}

【解答】

Head(L)=();

Tail(L)=(());

L的长度是2；
L的深度是2。

例题5-4　（北京航空航天大学1998年试题）

设$m\times n$阶稀疏矩阵A有t个非零元素，其三元组表表示为$LTMA[1:(t+1), 1:3]$，试问非零元素的个数t达到什么程度时用$LTMA$表示A才有意义？

【解答】$3(t+1)<m\times n$，因而$t<m\times n/3-1$时用$LTMA$表示A才有意义。

例题5-5　（东北大学1998年试题）

设有5对角矩阵，$A=(a_{ij})_{20\times 20}$，按特殊矩阵压缩存储的方式将其5条对角线上的元素存于数组$B[-10:m]$中，计算元素$A[15, 15]$的存储位置。

【解答】如果以行序为主，$A[15, 15]$为第$3+4+5\times 12+3=70$个，存储位置为$70-10-1=59$；
如果按对角线顺序，$A[15, 15]$为第$18+19+15=52$个，存储位置为$52-10-1=41$。

例题5-6　（北京邮电大学1998年试题）

已知广义表$L=((x, y, z), a, (u, t, w))$，从$L$表中取出原子项$t$的运算是_____。
供选择的答案：

A. head(tail(tail(L)))
B. tail(head(head(tail(L))))
C. head(tail(head(tail(L))))
D. head(tail(head(tail(tail(L)))))

【解答】

tail(L)=(a, (u, t, w))
tail(tail(L)=((u, t, w))
head(tail(tail(L)))=(u, t, w)
tail(head(tail(tail(L))))=(t, w)
head(tail(head(tail(tail(L))))=t

所以答案为D。

例题5-7　（北京邮电大学1998年试题）

将一个$A[1..100, 1..100]$的三对角矩阵，按行优先存入一维数组$B[1..298]$中，A中元素$A_{66,65}$（即该元素下标）在B数组中的位置k为_____。
供选择的答案：

A. 198　　B. 195　　C. 197

【解答】 B

例题5-8　（北京工业大学1998年试题）

试编写建立广义表存储结构的算法，要求在输入广义表的同时实现判断、建立。设广义表按如下形式输入（a_1，a_2，a_3，…，a_n）$n\geqslant 0$，其中a_i或为单字母表示的原子或为广义表，$n=0$时为只含空格字符的空表。

【解答】定义广义表存储结构：

```
TYPE list=^node;
        node=RECORD
             tag=0..1
             CASE
             0:(data:datatype);
                  1:(hp, tp:list);
             END;
```

算法如下：

```
PROCEDURE  create(VAR s1:list)
BEGIN
        read(x);    //此处字符必为空格、左括号或单个英文字母
    CASE x OF
        ' ':S1=NIL;    //若输入为空格；则置ls为空表
         '(':new(ls);ls^.tag:=1;create(ls^.hp);
         'a'-'z':new(ls);tag:=0;ls^.data:=x;
    END;
    read(x);
    IF ls<>NIL
        THEN IF X=',' THEN create(ls^.tp)
        ELSE ls^.next:=NIL;
END.
```

例题5-9 （中国科学院计算机网络信息中心2001年试题）

单选题：

设字符串S满足concat(head(S)，head(tail(tail(S))))=″ac″，(head和tail的定义同广义表)，则S=_____。

A. ″aabc″　　B. ″acbc″

C. ″accc″　　D. ″acac″

【解答】D

例题5-10 （中国科学院计算机网络信息中心2000年试题）

设A[1..100]是一个记录构成的数组，B[1..100]是一个整数数组，其值介于1至100之间，现要求按B[1..100]的内容调整A中记录的次序，比如当B[1]=11时，则要求将A[1]的内容调整到A[11]中去。规定可使用的附加空间为$O(1)$。

【解答】伪代码：

```
Rearrange(A, B){
    for i:=1 to 100 do {
        if B[i]≠i then{
```

```
                j:=B[i];temp=A[j];
                while j≠I do {
                     A[j]:=A[i];A[i]:=temp;j=B[j];
                     temp:=A[j];
                }
           }
           i:=i+1;
      }
}
```

例题5-11　（华北计算技术研究所2001年试题）

设广义表为A=(a，b，(c，d)，(e，(f，g)，(h，j)))，则下面式子的值为_____。

Head(Tail(Head(Tail(tail(A)))))

A. (g)　　B. (d)　　C. (e)　　D. (h)

【解答】A

例题5-12　（北京科技大学2002年试题）

设某单位职工工资表ST由“工资”、“扣除”和“实发金额”三项组成，其中工资项包括“基本工资”、“津贴”和“奖金”，扣除项包括“水”、“电”和“煤气”费用等。

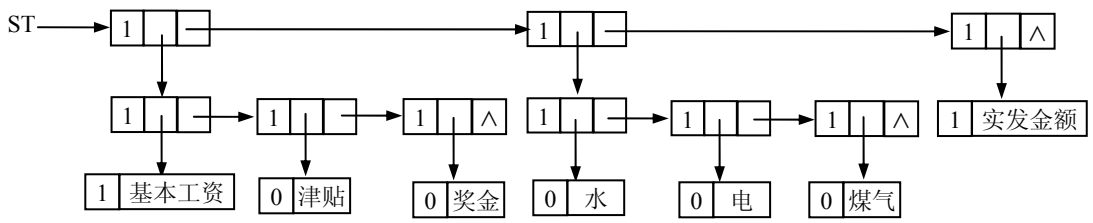
（1）请用广义表形式表示所描述的工资表ST，并用GetHead(ST)和GetTail(ST)函数提取表中的“奖金”项；

（2）用C语言描述广义表中的元素结构，并画出该工资表ST的存储结构。

【解答】

（1）ST=((基本工资，津贴，奖金)，(水，电，煤气)，实发金额)
GetHead(GetTail(GetTail(GetHead(ST))))=奖金

（2）
```
Struct node{ int tag;                /*标志域，1代表表结点，0代表元素结点*/
             struct node *hq，*tp;    /*表结点有效，指向表头和表尾的指针*/
             char data[10];          /*元素结点有效，元素值*/
  };
```



例题5-13　（武汉理工大学2002年试题）

选择题：

三对角线矩阵a[1..n][1..n]以行序为主顺序存储，其存储始址是b，每个元素占一个存储单元，则元素a[i][j]的存储始址为_____（1≤i，j≤n）。

供选择的答案：

A. b+2*j+i-2　　B. b+2*i+j-2　　C. b+2*j+i-3　　D. b+2*i+j-3

【解答】D

5.5 自 测 题

自测题5-1

编写一个过程，对一个$n\times n$矩阵，通过行变换，使其每行元素的平均值按递增顺序排列。

自测题5-2

编写子程序，将一维数组$A[1:n\times n]$($n\leqslant 10$)中的元素，按蛇形方式存放在二维数组$B[1:n, 1:n]$中。

即：$B[1, 1]=A[1]$，

$B[1, 2]=A[2]$，$B[2, 1]=A[3]$，

$B[3, 1]=A[4]$，$B[2, 2]=A[5]$，$B[1, 3]=A[6]$，

…

依次类推，如图5-2所示。

$$\begin{bmatrix} A[1] & A[2] & A[6] & A[7] & \Lambda \\ A[3] & A[5] & A[8] & A[14] & \Lambda \\ A[4] & A[9] & A[13] & \Lambda & \Lambda \\ A[10] & A[12] & \Lambda & \Lambda & \Lambda \\ A[11] & \Lambda & \Lambda & \Lambda & \Lambda \\ \Lambda & \Lambda & \Lambda & \Lambda & \Lambda \end{bmatrix}$$

图 5-2　二维数组 B

自测题5-3

判断正误：稀疏矩阵压缩存储后，必会失去随机存取功能。

自测题5-4

判断正误：若一个广义的表头为空表，则此广义表亦为空表。

自测题5-5

已知Head(Tail([Head(S)，Head(Tail(Tail(S)))]))=[a]，广义表S满足上式，则S为____。（其中，方括号表示广义表，圆括号表示函数。如[a，b]表示由a，b构成的广义表，而Head()表示取广义表的头部。）

A. [[a，b]，b，a]　　B. [[b，a]，[a]，[b]]　　C. [[a]，[a，b]，[b]]

D. [b，[a]，[a，b]]　E. [[a]，[b]，[b，a]]　　F. [[b]，[b，a]，[a]]

自测题5-6

画出下列广义表的存储结构图，并利用取表头和取表尾的操作分离出原子e。
$(a, (\ ), b), (((e)))$

自测题5-7

请画出下列广义表的图形表示。

（1）$D(A(), B(e), C(a, L(b, c, d)))$。
（2）$J_1(J_2, J_4(J_1, a, J_3(J_1)), J_3(J_1))$。

自测题5-8

若在矩阵$A_{m\times n}$中存在一个元素$a_{i-1,\ j-1}$，该元素是第i行元素中最小值且又是第j行元素中最大值，则称此元素为该矩阵的一个马鞍点。假设以二维数组存储矩阵$A_{m\times n}$，试设计一个求该矩阵所有马鞍点的算法，并分析你的算法在最坏情况下的时间复杂度。

自测题5-9

假设按低下标优先存储整数数组A(-3:8，3:5，-4:0，0:7)时，每一个元素的字节存储地址是100，每个整数占4个字节，问A(0，4，-2，5)的存储地址是什么？

5.6　自测题答案

【自测题5-1】

```
Trans(mat, no)
float  *mat                  /*matrix[no][no]*/
int no;
{
     int i, j, k, l;
     float sum, minf;
     float *p, *pt, *pk, *pi;

     p=(float *)malloc(no*sizeof(float));
     for(i=0;i<no;i++){          /*sum up each row */
          sum=0.0;pt=mat+i*no;
          for(j=0;j<no;j++){
               pt++;sum+=(pt);
          }
          *(p+i)=sum;
     }
     for(i=0;i<no-1;j++){
          minf=*(p+i);k=i;
          for(j=i+1;j<no;j<no;j++)/*seek for the index of minimum */
               if(minf>p[j])  {
                    k=j;
```

```
                minf=p[j];
            }
        if(k!=i){          /*exchange the rows */
            pk=mat+(no*k);
            pi=mat+(no*i);
            for(l=0;l<no;l++){  /*row k exchanges with row i*/
                sum=*(pi+1);
                *(pi+1)=*(pk+1);
                *(pk+1)=*sum;
            }
            sum=p[i];
            p[i]=p[k];
            p[k]=sum;          /*exchange p[i]with p[k] */
            }
        }
        free(p);
}
```

【自测题5-2】

```
PROCEDURE Yh(A:ARRAY[1..n*n]OF integer;
            VAR B:ARRAY[1..n, 1..n]OF integer);
VAR
        i, j, n, k, L, LL,  m:integer;
BEGIN
        m:=1;
        FOR k:=1 TO 2*n-1 DO
        BEGIN
            IF k<n THEN LL:=k;
                ELSE LL:=2*n-k;
            FOR L:=1 TO LL DO
            BEGIN
                IF(k MOD 2<>0)THEN
                BEGIN
                    i:=LL-L+1;
                    j:=L;
                END
            ELSE
            BEGIN
                i:=L;
                j:=LL-L+1;
            END
            IF(k>=n)THEN
            BEGIN
                i:=i+n-LL;
                j:=j+n-LL;
            END
            B[i, j]:=A[m];
            m:=m+1;
        END
    END
```

END;

【自测题5-3】

正确。

【自测题5-4】

例如，广义表*L*=((), (*A*, *B*))的表头为空表，但*L*不是空表。错误。

【自测题5-5】

可用代入法求解。答案为F。

【自测题5-6】

存储空间如图5-3所示。

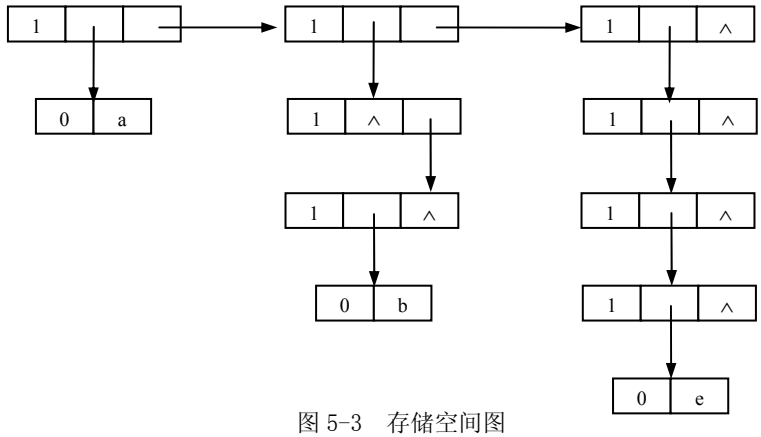


图 5-3　存储空间图

设*A*=(*a*, (), *b*), (((*e*))),

Head(Head(Head(Head(Tail(Tail(*A*))))))=*e*。

【自测题5-7】

（1）*D*(*A*(), *B*(*e*), *C*(*a*, *L*(*b*, *c*, *d*))的存储结构如图5-4所示。

错误！

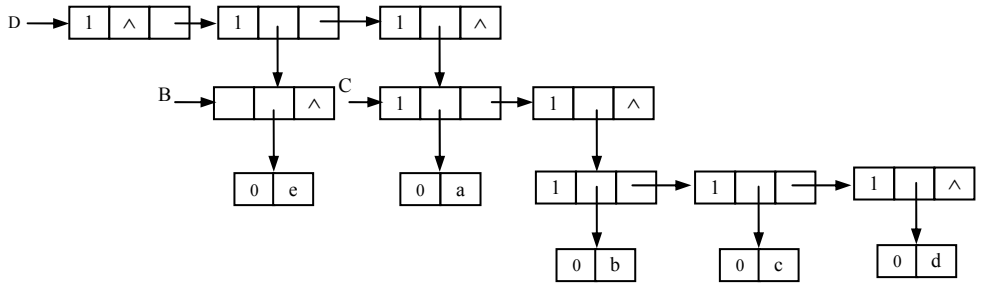


图 5-4　*D*(*A*(), *B*(*e*), *C*(*a*, *L*(*b*, *c*, *d*))的存储结构

（2）$J_1(J_2, J_4(J_1, a, J_3(J_1)), J_3(J_1))$的存储结构如图5-5所示。

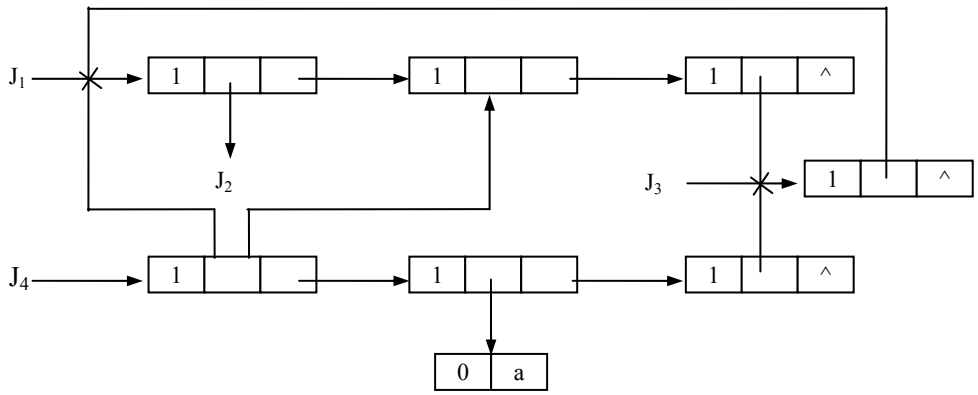


图 5-5　$J_1(J_2, J_4(J_1, a, J_3(J_1)), J_3(J_1))$的存储结构

【自测题5-8】

【分析】设定两个数组　　$max[0..n-1]$：第j行最大值所在行号

$min[0..m-1]$：第i行最小值所在列号

则　第j列最大值为$A[max[j], j]$

第i行最小值为$A[i, min[i]]$

通过二重循环找到每一列的最大值和每一行的最小值，记录在max和min数组中。扫描max和min数组，找到马鞍点。

算法如下：

```
for i=0 to n-1 do
   max[i]=0;
for i=0 to m-1 do
   min[i]=0;
FOR i=0 TO m-1
     FOR j=0 TO n-1 {
          IF (A[i, j]>A[max[j], j])
               max[j]=i;
          IF (A[i, j]<A[i, min[i]])
               min[i]=j;
     }
     FOR i=0 TO m=1 {
          j=min[i];
          IF(i==max[j])
               print(马鞍点A[i, j]);
     }
```

时间复杂度为$O(mn)$。

【自测题5-9】

n维数组的数据元素$(j_1, j_2, \cdots, j_n)$存储位置的公式如下：

$$LOC(j_1, j_2, \cdots, j_n)=LOC(0, 0, 000, 0)+(b_2\times\cdots\times b_n\times j_1+b_3\times\cdots\times b_n\times j_{n-1})L$$

$$=LOC(0，0，\cdots，0)+(\sum_{i=1}^{n-1} j_i \prod_{k=i+1}^{n} b_k + j_n)L$$

其中b_1，b_2，…，b_n为各维的长度，每个数据元素占L个存储单元。

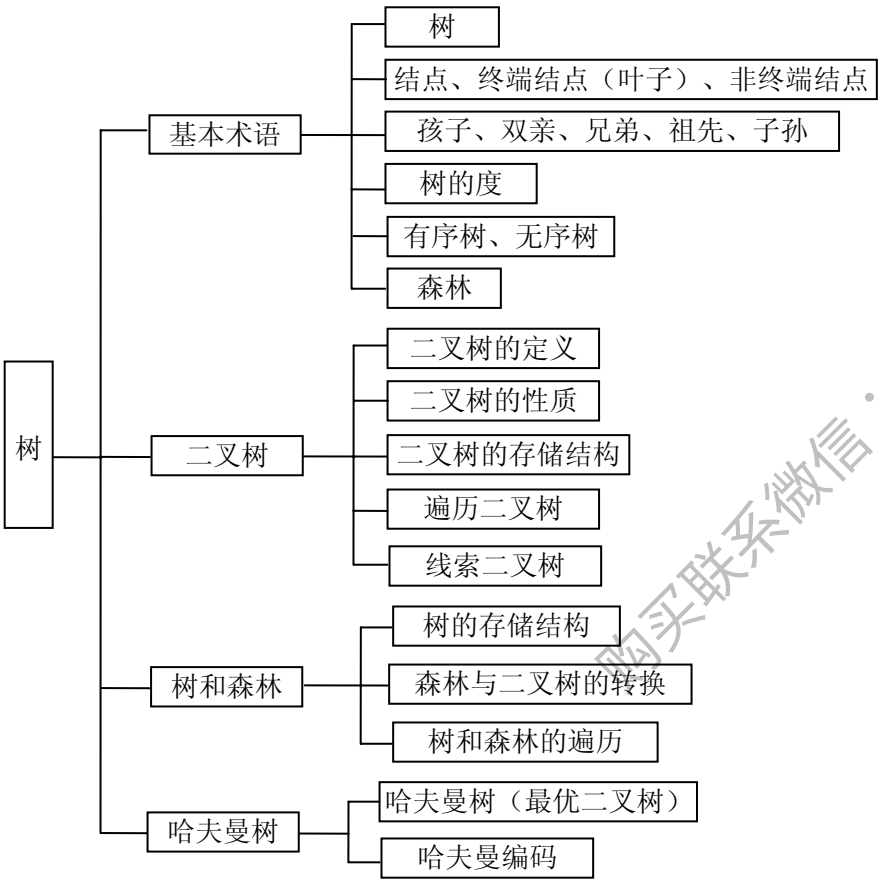
答案：

规范化地在数组$A(0:11，0:2，0:4，0:7)$中求$A(3，1，2，5)$的存储位置：

$$L(3，1，2，5)=100+4\times(3\times2\times4\times7+1\times4\times7+2\times7+5)=1164$$

第 6 章　树和二叉树

6.1　基本知识结构图



6.2　知 识 点

1. 基本概念

树是n个结点的有限集。在任意一棵非空树中：（1）有且仅有一个特定的称为根的结点；（2）当$n>1$时，其余结点可分为$m(m>0)$个互不相干的有限集T_1，T_2，…，T_m，其中每一

个集合本身又是一棵树，并且称为根的子树。树的定义是一种递归定义。

树的结点包括一个数据元素及若干指向其子树的分支。结点拥有的子树数称为结点的度。度为0的结点称为终端结点（叶子）。度不为0的结点称为非终端结点或分支结点。

树的度是树内各结点的度的最大值。

结点的子树的根称为该结点的孩子，该结点称为孩子的双亲。同一个双亲的孩子之间互称兄弟。从根到该结点所经分支上的所有结点称该结点的祖先。以某结点为根的子树中的任意结点都称为该结点的子孙。

树中结点的各子树从左至右是有次序的，则称该树为有序树，否则称为无序树。

森林是m($m\geqslant 0$)棵互不相交的树的集合。

二叉树是另一种树型结构，它的特点是每个结点至多只有两棵子树，并且，二叉树的子树有左右之分，其次序不能任意颠倒。

2. 二叉树的性质和存储结构

二叉树的性质：

- 在二叉树的第i层上至多有2^{i-1}个结点（$i\geqslant 1$）。
- 深度为k的二叉树至多有2^k-1个结点（$k\geqslant 1$）。
- 对任何一棵二叉树T，如果其终端结点数为n_0，度为2的结点数为n_2，则$n_0=n_2+1$。
- 具有n个结点的完全二叉树的深度为$\lfloor \log_2 n \rfloor+1$。
- 如果对一棵有n个结点的完全二叉树的结点按层序编号（从第1层到第$\lfloor \log_2 n \rfloor+1$层，每层从左到右），则对任一结点$i$（$1\leqslant i\leqslant n$），有：

  ① 如果i=1，则结点i是二叉树的根，无双亲；若i>1，则其双亲PARENT(i)是结点$\lfloor i/2 \rfloor$。

  ② 如果2i>n，则结点i无左孩子；否则其左孩子LCHILD(i)是结点2i。

  ③ 如果2i+1>n，则结点i无右孩子；否则其右孩子RCHILD(i)是结点2i+1。

3. 二叉树的存储结构

二叉树的顺序存储是用一组地址连续的存储单元依次自上而下、自左至右存储完全二叉树上的结点元素。对于一般二叉树，则应将其每个结点与完全二叉树上的结点相对照，存储在一维数组的相应分量中，以“0”表示不存在的结点。

在二叉树的链式存储结构中，结点至少包含3个域：数据域和左、右指针域。左、右指针分别指向其左右子树的两个分支。有时，为了便于找到结点的双亲，则还可在结点结构中增加一个指向其双亲结点的指针域。

4. 遍历二叉树和线索二叉树

遍历二叉树是按某条搜索路径巡访树中每个结点，使得每个结点均被访问一次，而且仅被访问一次。有先（根）序遍历、中（根）序遍历和后（根）序遍历三种方案。

利用二叉树中的空链域指向结点的前驱和后继的指针叫做线索，加上线索的二叉树称为线索二叉树。对二叉树以某种顺序遍历使其变为线索二叉树的过程称为线索化。

5．哈夫曼树

从树中一个结点到另一个结点之间的分支构成这两个结点之间的路径，路径上的分支数目称为路径长度，树的路径长度是从树根到每一个结点的路径长度之和。假设有n个权值$\{w_1, w_2, \cdots, w_n\}$，构造一棵有$n$个叶子结点的二叉树，每个叶子结点带权为$w_i$，则其中带权路径长度最小的二叉树称为最优二叉树或哈夫曼树。

6．哈夫曼编码

哈夫曼编码是哈夫曼树的一个应用。在快速远距离通信中要对字符进行编码，可以根据字符出现的次数设计对每个字符编码的长度，并且任一个字符的编码都不是另一个字符编码的前缀（这种编码称前缀编码），这样可使传送的电文尽可能短且有惟一的译码。以n种字符出现的频率作权，设计一棵哈夫曼树，由此得到的二进制前缀编码称为哈夫曼编码。

6.3　习题及参考答案

1．已知一棵树边的集合为{<*I*，*M*>，<*I*，*N*>，<*E*，*I*>，<*B*，*E*>，<*B*，*D*>，<*A*，*B*>，<*G*，*J*>，<*G*，*K*>，<*C*，*G*>，<*C*，*F*>，<*H*，*L*>，<*C*，*H*>，<*A*，*C*>}，请画出这棵树，并回答下列问题：

（1）哪个是根结点？
（2）哪些是叶子结点？
（3）哪个是结点G的双亲？
（4）哪些是结点G的祖先？
（5）哪些是结点G的孩子？
（6）哪些是结点E的子孙？
（7）哪些是结点E的兄弟？哪些是结点F的兄弟？
（8）结点B和N的层次号分别是什么？
（9）树的深度是多少？
（10）以结点C为根的子树的深度是多少？

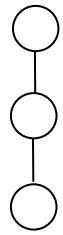
【解答】（1）A　　（2）D，M，N，F，J，K，L
（3）C　　（4）A，C　　（5）J，K
（6）I，M，N　　（7）E的兄弟是D，F的兄弟是G和H
（8）2，5　　（9）5
（10）3

2．一棵度为2的树与一棵二叉树有何区别？

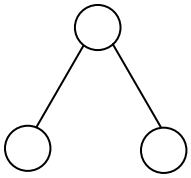
【解答】二叉树的子树有左右之分，其次序不能任意颠倒。

3．试分别画出具有3个结点的树和3个结点的二叉树的所有不同形态。

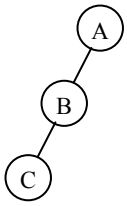
【解答】含三个结点的树只有两种形态：（1）和（2）；而含3个结点的二叉树可能有下列5种形态：（一），（二），（三），（四），（五）。



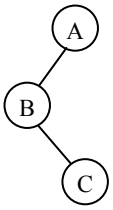
（1）



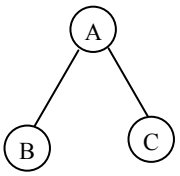
（2）



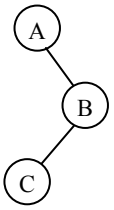
（一）



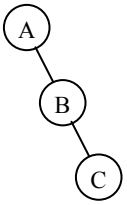
（二）



（三）



（四）



（五）

4．一棵深度为H的满k叉树有如下性质：第H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开始对全部结点编号，问：

（1）各层的结点数目是多少？

（2）编号为p的结点的父结点（若存在）的编号是多少？

（3）编号为p的结点的第i个儿子结点（若存在）的编号是多少？

（4）编号为p的结点有右兄弟的条件是什么？其右兄弟的编号是多少？

【解答】（1）第i层有k^{i-1}个结点；

（2）p=1时，该结点为根，无父结点；

否则其父结点编号为$\left\lfloor \dfrac{P+(k-2)}{k} \right\rfloor$（$k \geqslant 2$）。

（3）其第k-1个儿子的编号为$p \cdot k$。所以，如果它有儿子，则其第i个儿子的编号为$p \cdot k+(i-(k-1))$；

（4）$(p-1)\%k \neq 0$时，该结点有右兄弟，其右兄弟的编号为p+1。

5．已知一棵度为k的树中有n_1个度为1的结点，n_2个度为2的结点，…，n_k个度为k的结点，问该树中有多少个叶子结点？

【解答】$n_0=1+\sum_{i=1}^{k}(i-1)n_i$

6．已知在一棵含有n个结点的树中，只有度为k的分支结点和度为0的叶子结点。试求该树含有叶子结点的数目。

【解答】$n-\frac{n-1}{k}$

7．一棵含有n个结点的k叉树，可能达到的最大深度和最小深度各为多少？

【解答】显然，能达到最大深度的是单支树，其深度为n；深度最小的是完全k叉树。

8．在二叉树的顺序存储结构中，实际上隐含着双亲的信息，因此可和三叉链表对应。假设每个指针域占4个字节的存储，每个信息域占k个字节的存储。试问：对于一棵有n个结点的二叉树，且在顺序存储结构中最后一个结点的下标为m，在什么条件下顺序存储结构比三叉链表更节省空间？

【解答】这个条件是$k<\frac{12n}{m-n}$。

思考：一棵一般的二叉树存于顺序结构上时，应如何判别某个结点是否为叶子结点和如何判别非结点分量。

9．对题3所得各种形态的二叉树，分别写出前序、中序和后序遍历的序列。

【解答】

	（一）	（二）	（三）	（四）	（五）
前序	ABC	ABC	ABC	ABC	ABC
中序	CBA	BCA	BAC	ACB	ABC
后序	CBA	CBA	BCA	CBA	CBA

10．假设n和m为二叉树中两结点，用“1”、“0”或“ϕ”（分别表示肯定、恰恰相反或者不一定）填写下表。

问 / 答 / 已知	前序遍历时 n在m前？	中序遍历时 n在m前？	后序遍历时 n在m前？
n在m左方			
n在m右方			
n是m祖先			
n是m子孙			

注：如果（1）离a和b最近的共同祖先p存在，且（2）a在p的左子树中，b在p的右子树中，则称a在b的左方（即b在a的右方）。

【解答】解此类题应画图帮助思考。

1	1	1
0	0	0
1	ϕ	0
0	ϕ	1

11．找出所有满足下列条件的二叉树：

（a）它们在先序遍历和中序遍历时，得到的结点访问序列相同；

（b）它们在后序遍历和中序遍历时，得到的结点访问序列相同；

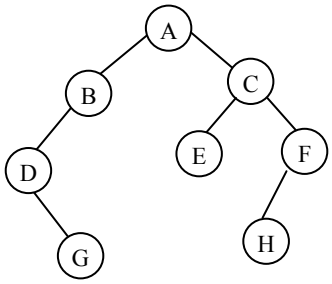
（c）它们在先序遍历和后序遍历时，得到的结点访问序列相同。

【解答】（a）任意一个结点都只有右子树或者为叶子结点；

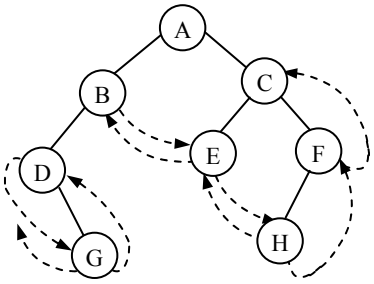
（b）任意一个结点都只有左子树或者为叶子结点；

（c）只有根结点。

12．请对下图所示二叉树进行后序线索化，为每个空指针建立相应的前驱或后继线索。



【解答】



13．将下列二叉链表改为先序线索链表（不画出树的形态）。

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Info	A	B	C	D	E	F	G	H	I	J	K	L	M	N
Ltag														
Lchild	2	4	6	0	7	0	10	0	12	13	0	0	0	0
Rtag														
Rchild	3	5	0	0	8	9	11	0	0	0	14	0	0	0

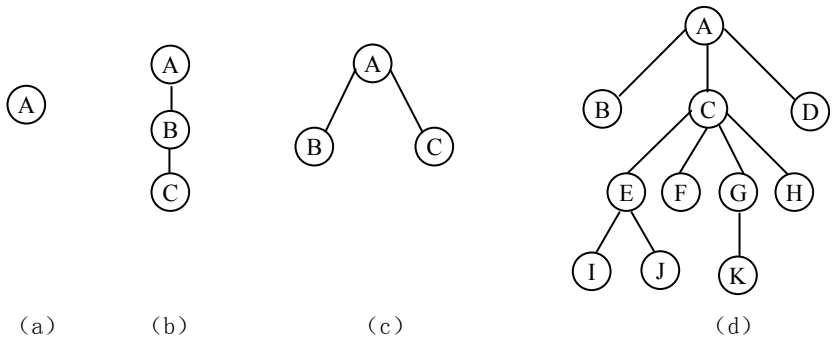
【解答】

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Info	A	B	C	D	E	F	G	H	I	J	K	L	M	N
Ltag	0	0	0	1	0	1	0	1	0	0	1	1	1	1
Lchild	2	4	6	0	7	0	10	0	12	13	0	0	0	0
Rtag	0	0	1	1	0	0	0	1	1	1	0	1	1	1
Rchild	3	5	0	0	8	9	11	0	0	0	14	0	0	0

14．试讨论，能否在一棵中序全线索二叉树上查找给定结点* p在后序序列中的后继。

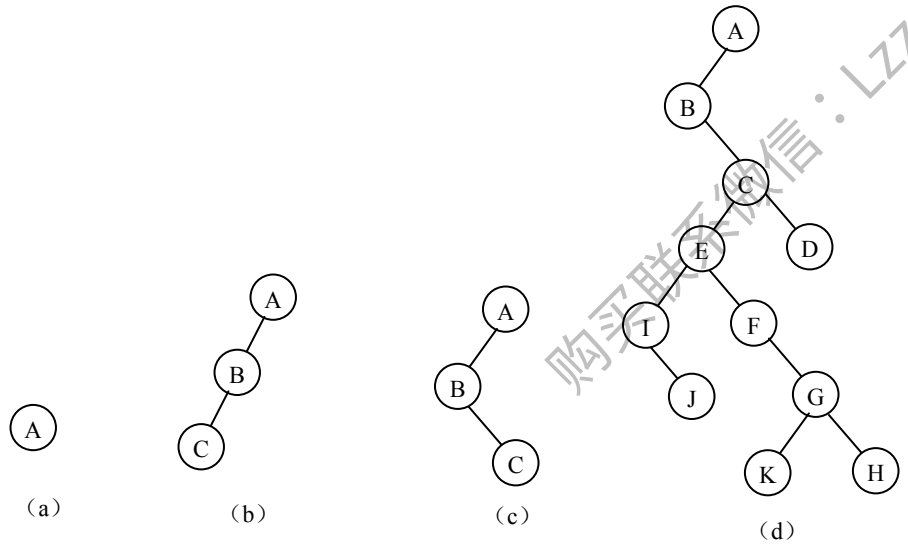
【解答】本题难点在于找当前结点*p的双亲结点，由于二叉树以中序全线索链表表示，则以结点*p为根的子树中必存在这样一个结点：其前驱或后继线索指向*p的双亲结点。

15．分别画出和下列树对应的各个二叉树；



（a）　（b）　（c）　（d）

【解答】



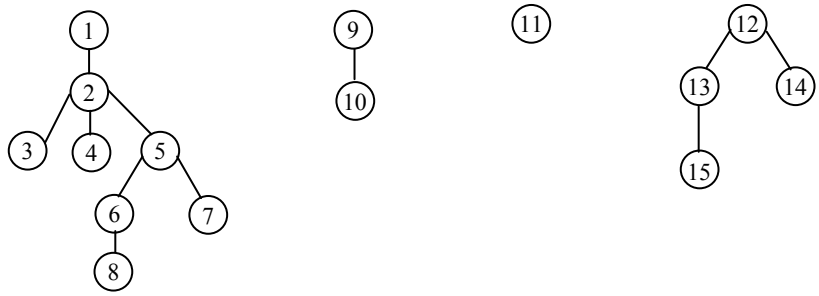
（a）　（b）　（c）　（d）

16．将下列森林转换为相应的二叉树，并分别按以下说明进行线索化：

（1）先序前驱线索化；

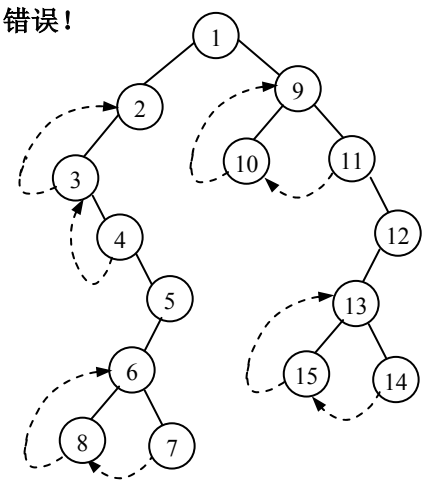
（2）中序全线索化前驱线索和后继线索；

（3）后序后继线索化。

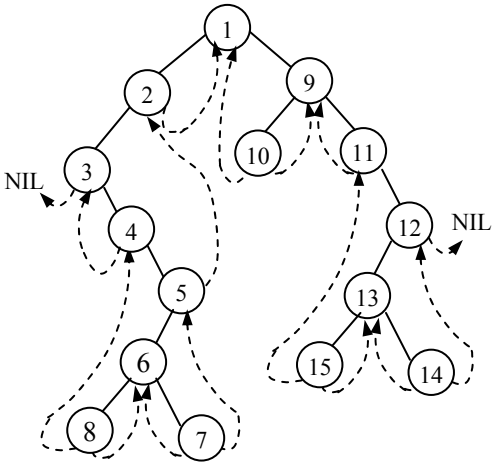


【解答】

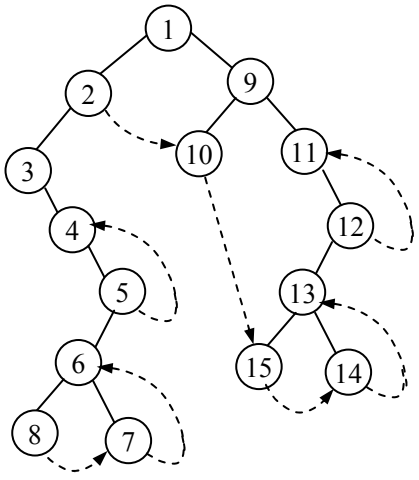
错误！



（1）

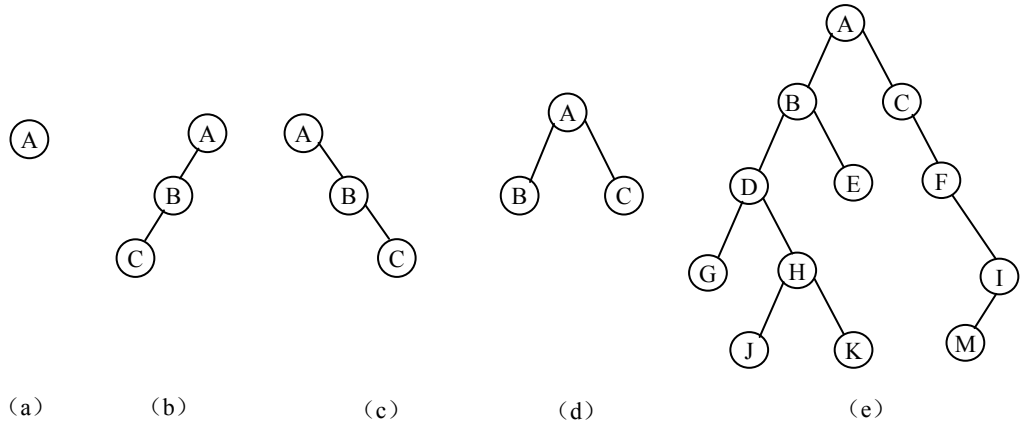


（2）



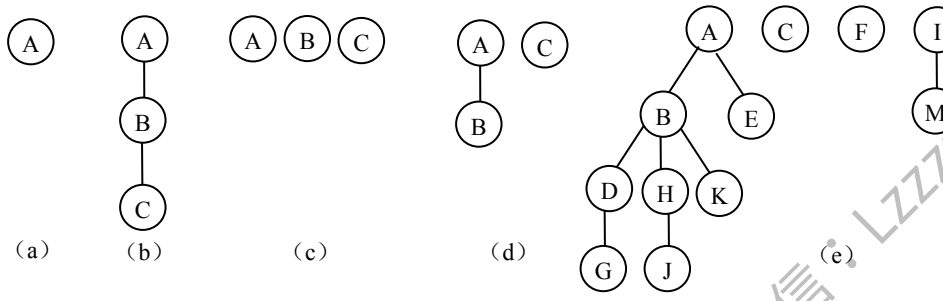
（3）

17．画出和下列二叉树相应的森林：



（a）　（b）　（c）　（d）　（e）

【解答】



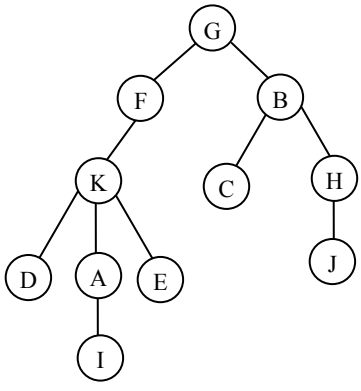
（a）　（b）　（c）　（d）　（e）

18．对于15题中给出的各树分别求出以下遍历序列：
（1）先根序列；（2）后根序列。

【解答】（1）（a）A　（b）ABC　（c）ABC　（d）ABCEIJFGKHD
（2）（a）A　（b）CBA　（c）BCA　（d）BIJEFKGHCDA

19．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；
树的后根次序访问序列为DIAEKFCJHBG。

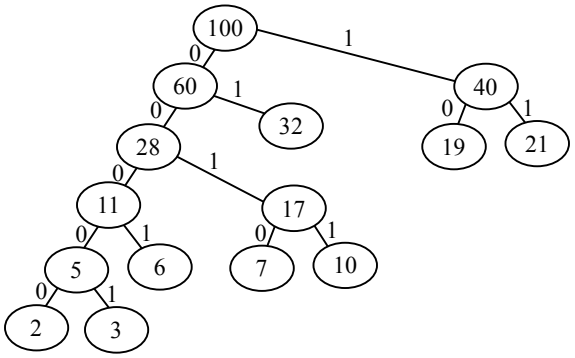
【解答】



20．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。

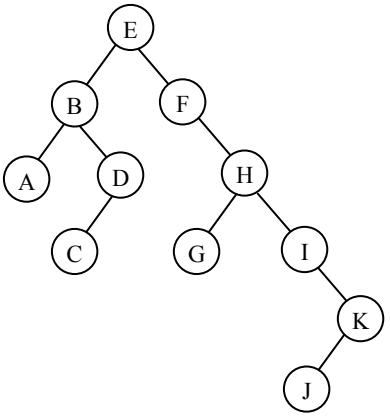
【解答】哈夫曼编码方案的带权路径长度为WPL_{HF}=2.61，而等长编码的路径长度为WPL_{EQ}=3，显然前者可大大提高通信信道的利用率，提高报文发送速度或/和节省存储空间。下面给出两种编码的对照表及哈夫曼树的逻辑结构。

频数	7	19	2	6	32	3	21	10
哈夫曼编码	0010	10	00000	0001	01	00001	11	011
等长编码	000	001	010	011	100	101	110	111



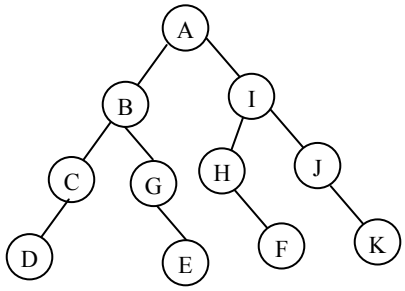
21．假设一棵二叉树的先序序列为EBADCFHGIKJ和中序序列为ABCDEFGHIJK。请画出该树。

【解答】



22．假设一棵二叉树的中序序列为DCBGEAHFIJK和后序序列为DCEGBFHKJIA。请画出该树。

【解答】



23. 假定用两个一维数组L[1..n]和R[1..n]作为有n个结点的二叉树的存储结构，L[i]和R[i]分别指示结点i的左孩子和右孩子，0表示空。试写一个算法判别结点u是否为结点v的子孙。

【分析】存储结构可以认为是静态二叉链表。u是v的子孙 ⇔ v是u的祖先。如果在一棵给定的二叉树上判别两个结点之间的这种关系是经常性操作，则第一次建立T后，以后的操作可大大节省时间。进而推广为更一般的结论：如果一个给定数据结构不适于所要求的某种操作，而此数据结构已经在一个软件系统中使用，则不改数据结构，而采取这种“向上兼容”的扩充策略是一种可以考虑的方法，当然要注意维护数据结构的一致性，例如，插/删操作。

【解答】算法如下：

```
int Is_Descendant_C(int u, int v)//在孩子存储结构上判断u是否为v的子孙，是则返回1，否则返回0
{
  if(u==v) return 1;
  else
  {
    if(L[v])
      if (Is_Descendant(u, L[v])) return 1;
    if(R[v])
      if (Is_Descendant(u, R[v])) return 1; //这是个递归算法
  }
  return 0;
}//Is_Descendant_C
```

在以下24~26题和29~41题中，均以二叉链表作为二叉树的存储结构。

24. 若已知两棵二叉树B1和B2皆为空，或者皆不空且B1的左、右子树和B2的左、右子树分别相似，则称二叉树B1和B2相似。试编写算法，判别给定两棵二叉树是否相似。

【解答】算法如下：

```
int Bitree_Sim(Bitree B1, Bitree B2)//判断两棵树是否相似的递归算法
{
  if(!B1&&!B2) return 1;
  else if(B1&&B2&&Bitree_Sim(B1->lchild, B2->lchild)&&
        Bitree_Sim(B1->rchild, B2->rchild))
          return 1;
```

```
      else return 0;
}//Bitree_Sim
```

25．试利用栈的基本操作写出先序遍历的非递归形式的算法。

【解答】算法如下：

```
void PreOrder_Nonrecursive(Bitree T)//先序遍历二叉树的非递归算法
{
  InitStack(S);
  Push(S, T); //根指针进栈
  while(!StackEmpty(S))
  {
    while(Gettop(S, p)&&p)
    {
      visit(p->data);
      push(S, p->lchild);
    } //向左走到尽头
    pop(S, p);
    if(!StackEmpty(S))
    {
      pop(S, p);
      push(S, p->rchild); //向右一步
    }
  }//while
}//PreOrder_Nonrecursive
```

26．同上一题条件，写出后序遍历的非递归算法（提示：为分辨后序遍历时两次进栈的不同返回点，需在指针进栈时同时将一个标志进栈）。

【分析】为了区分两次过栈的不同处理方式，在堆栈中增加一个mark域，mark=0表示刚刚访问此结点，mark=1表示左子树处理结束返回，mark=2表示右子树处理结束返回，每次根据栈顶元素的mark域值决定做何种动作。

【解答】算法如下：

```
typedef struct {
                BTNode* ptr;
                enum {0, 1, 2} mark;
            } PMType; //有mark域的结点指针类型
void PostOrder_Stack(BiTree T)//后续遍历二叉树的非递归算法，用栈
{
  PMType a;
  InitStack(S); //S的元素为PMType类型
  Push (S, {T, 0}); //根结点入栈
  while(!StackEmpty(S))
  {
      Pop(S, a);
      switch(a.mark)
      {
         case 0:
            Push(S, {a.ptr, 1}); //修改mark域
```

```
            if(a.ptr->lchild) Push(S, {a.ptr->lchild, 0}); //访问左子树
            break;
        case 1:
            Push(S, {a.ptr, 2}); //修改mark域
            if(a.ptr->rchild) Push(S, {a.ptr->rchild, 0}); //访问右子树
            break;
        case 2:
            visit(a.ptr); //访问结点，返回
    }
  }//while
}//PostOrder_Stack
```

27. 假设在二叉链表的结点中增设两个域：双亲域（parent）以指示其双亲结点；标志域（mark取值0..2）以区分在遍历过程中到达该结点时应继续向左或向右或访问该结点。试以此存储结构编写不用栈进行后序遍历的递推形式的算法。

【分析】建立一个当前结点所处状态的概念，结点的标志域正是起一个区分当前结点正处在什么状态的作用。当前结点的状态可能为：（1）由其双亲结点转换来；（2）由其左子树遍历结束转换来；（3）由其右子树遍历结束转换来。则算法主要循环体的每一次执行都按当前点的状态进行处理，或切换当前结点，或切换当前结点的状态。

【解答】算法如下：

```
void PostOrder (BiTree root)
{
    //设二叉树的结点中含有四个域：mark， parent， lchild， rchild。其中，
    //mark域的初值均为零，指针root指向根结点，后序遍历此二叉树。
    p=root;
    while (p)
        switch (p->mark ) {
            case 0：
                p->mark=1;
                if (p->lchild) p=p->lchild;
                break;
            case 1：
                p->mark=2;
                if (p->rchild) p=p->rchild;
                break;
            case 2：
                p->mark=0;
                visit (*p) ;
                p=p->parent;
                break;
            default：;
    }
} //PostOrder
```

通过修改条件可以使本题继续增加难度：（1）mark的值只取0或1；（2）去掉双亲指针域，用逆转链的方法维持双亲结点指针。

28．若在二叉链表的结点中只增设一个双亲域以指示其双亲结点，则在遍历过程中能否不设栈？试以此存储结构编写不设栈进行中序遍历的递推形式的算法。

【解答】算法如下：

```
typedef struct {
                int data;
                PBTNode *lchild;
                PBTNode *rchild;
                PBTNode *parent;
            } PBTNode, PBitree; //有双亲指针域的二叉树结点类型
void Inorder_Nonrecursive(PBitree T)//不设栈非递归遍历有双亲指针的二叉树
{
    p=T;
    while(p->lchild) p=p->lchild; //向左走到尽头
    while(p)
    {
        visit(p);
        if(p->rchild) //寻找中序后继:当有右子树时
        {
            p=p->rchild;
            while(p->lchild) p=p->lchild; //后继就是在右子树中向左走到尽头
        }
        else if(p->parent->lchild==p) p=p->parent; //当自己是双亲的左孩子时后继就是双亲
           else
           {
               p=p->parent;
               while(p->parent&&p->parent->rchild==p) p=p->parent;
               p=p->parent;
           } //当自己是双亲的右孩子时后继就是向上返回直到遇到自己是在其左子树中的祖先
      }//while
}//Inorder_Nonrecursive
```

29．编写递归算法，在二叉树中求位于先序序列中第k个位置的结点的值。

【解答】算法如下：

```
int c, k; //这里把k和计数器c作为全局变量处理
void Get_PreSeq(Bitree T)//求先序序列为k的结点的值
{
    if(T)
    {
        c++; //每访问一个子树的根都会使前序序号计数器加1
        if(c==k)
        {
            printf("Value is %d\n", T->data);
            exit (1);
        }
        else
        {
            Get_PreSeq(T->lchild); //在左子树中查找
```

```
            Get_PreSeq(T->rchild); //在右子树中查找
        }
    }//if
}//Get_PreSeq
main()
{
    ...
    scanf("%d", &k);
    c=0; //在主函数中调用前，要给计数器赋初值0
    Get_PreSeq(T, k);
    ...
}//main
```

30．编写递归算法，计算二叉树中叶子结点的数目。

【解答】算法如下：

```
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
```

31．编写递归算法，将二叉树中所有结点的左、右子树相互交换。

【解答】算法如下：

```
void Bitree_Revolute(Bitree T)//交换所有结点的左右子树
{
    T->lchild<->T->rchild; //交换左右子树
    if(T->lchild) Bitree_Revolute(T->lchild);
    if(T->rchild) Bitree_Revolute(T->rchild); //左右子树再分别交换各自的左右子树
}//Bitree_Revolute
```

32．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。

【解答】算法如下：

```
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
```

33．编写递归算法：对于二叉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。

【分析】建议由两个算法实现，即释放被删子树上所有结点空间可单独写一个算法。注意易犯的错误是参数的设置不适当。

【解答】算法如下：

```
void Del_Sub_x(Bitree T, int x)//删除所有以元素x为根的子树
{
    if(T->data==x) Del_Sub(T); //删除该子树
    else
    {
        if(T->lchild) Del_Sub_x(T->lchild, x);
        if(T->rchild) Del_Sub_x(T->rchild, x); //在左右子树中继续查找
    }//else
}//Del_Sub_x
void Del_Sub(Bitree T)//删除子树T
{
    if(T->lchild) Del_Sub(T->lchild);
    if(T->rchild) Del_Sub(T->rchild);
    free(T);
}//Del_Sub
```

34．编写复制一棵二叉树的非递归算法。

【解答】算法如下：

```
void Bitree_Copy_Nonrecursive(Bitree T, Bitree &U)//非递归复制二叉树
{
    InitStack(S1);InitStack(S2);
    push(S1, T); //根指针进栈
    U=(BTNode*)malloc(sizeof(BTNode));
    U->data=T->data;
    q=U;push(S2, U);
    while(!StackEmpty(S))
    {
        while(Gettop(S1, p)&&p)
        {
            q->lchild=(BTNode*)malloc(sizeof(BTNode));
            q=q->lchild;q->data=p->data;
```

```
            push(S1, p->lchild);
            push(S2, q);
        } //向左走到尽头
        pop(S1, p);
        pop(S2, q);
        if(!StackEmpty(S1))
        {
            pop(S1, p);pop(S2, q);
            q->rchild=(BTNode*)malloc(sizeof(BTNode));
            q=q->rchild;q->data=p->data;
            push(S1, p->rchild); //向右一步
            push(S2, q);
        }
    }//while
}//BiTree_Copy_Nonrecursive
```

35. 编写按层次顺序（同一层自左至右）遍历二叉树的算法。

【分析】按层次遍历是基于另一种搜索策略的遍历，它的原则是先被访问的结点的左、右孩子结点先被访问，因此在遍历过程中需利用具有先进先出特性的队列。

【解答】算法如下：

```
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
```


36. 已知在二叉树中，*root为根结点，*p和*q为二叉树中两个结点，试编写求距离它们最近的共同祖先的算法。

【分析】此题不宜写成递归算法。注意考察非递归遍历算法中栈的状态。

【解答】算法如下：

```
int found=FALSE;
Bitree* Find_Near_Ancient(Bitree T, Bitree p, Bitree q)//求二叉树T中结点p和q的最近共同祖先
{
    Bitree pathp[ 100 ], pathq[ 100 ] //设立两个辅助数组暂存从根到p, q的路径
    Findpath(T, p, pathp, 0);
    found=FALSE;
    Findpath(T, q, pathq, 0); //求从根到p, q的路径放在pathp和pathq中
    for(i=0;pathp[i]==pathq[i]&&pathp[i];i++); //查找两条路径上最后一个相同结点
      return pathp[--i];
```

```
}//Find_Near_Ancient
void Findpath(Bitree T, Bitree p, Bitree path[ ], int i)//求从T到p路径的递归算法
{
    if(T==p)
    {
        found=TRUE;
        return; //找到
    }
    path[i]=T; //当前结点存入路径
    if(T->lchild) Findpath(T->lchild, p, path, i+1); //在左子树中继续寻找
    if(T->rchild&&!found) Findpath(T->rchild, p, path, i+1); //在右子树中继续寻找
    if(!found) path[i]=NULL; //回溯
}//Findpath
```

37. 编写算法判别给定二叉树是否为完全二叉树。

【分析】基于按层次顺序遍历的搜索策略较为适宜。也可以另设一个布尔数组记录访问过的结点，最后检查数组中是否只有一个TRUE平台，但这样做的时空效率低一些。若读者希望写一个递归算法，则先要给出完全二叉树的与原定义等价的递归定义。然后设计一个判别给定二叉树是满二叉树、不满的完全二叉树还是非完全二叉树的递归函数。

【解答】算法如下：

```
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
```

38. 假设以三元组（F，C，L/R）的形式输入一棵二叉树的诸边（其中F表示双亲结点的标识，C表示孩子结点标识，L/R表示C为F的左孩子或右孩子），且在输入的三元组序列中，C是按层次顺序出现的。设结点的标识是字符类型。F='^'时C为根结点标识，若C也为'^'，则表示输入结束。例如，如下二叉树的三元组序列输入格式为：

```
^AL
ABL
ACR
```

BDL
CEL
CFR
DGR
FHL
^ ^ L

试编写算法，由输入的三元组序列建立二叉树的二叉链表。

【分析】按层次顺序遍历一棵逻辑上的二叉树，“访问当前结点”的操作为建立该结点和给对应域赋值。

【解答】算法如下：

```
Status CreateBitree_Triplet(Bitree &T)//输入三元组建立二叉树
{
    if(getchar()!='^') return ERROR;
    T=(BTNode*)malloc(sizeof(BTNode));
    p=T;p->data=getchar();
    getchar(); //滤去多余字符
    InitQueue(Q);
    EnQueue(Q, T);
    while((parent=getchar())!='^'&&(child=getchar())&&(side=getchar()))
    {
        while(QueueHead(Q)!=parent&&!QueueEmpty(Q)) DeQueue(Q, e);
        if(QueueEmpty(Q)) return ERROR; //未按层序输入
        p=QueueHead(Q);
        q=(BTNode*)malloc(sizeof(BTNode));
        if(side=='L') p->lchild=q;
        else if(side=='R') p->rchild=q;
            else return ERROR; //格式不正确
            q->data=child;
        EnQueue(Q, q);
    }
    return OK;
}//CreateBitree_Triplet
```

39. 编写一个算法，输出以二叉树表示的算术表达式，若该表达式中含有括号，则在输出时应添上。

【分析】由于从表达式建立二叉树的过程是后序遍历的过程，因此建成的二叉树惟一确定了表达式的求值过程，即其左、右子树根的运算先于根的运算进行。若左子树根运算符的优先树小于根运算符的优先数，或右子树根运算符的优先数不大于根运算符的优先数时，则在原表达式中必存在括弧。

【解答】算法如下：

```
Status Print_Expression(Bitree T)//按标准形式输出以二叉树存储的表达式
{
    if(T->data是字母) printf("%c", T->data);
```

```
    else if(T->data是操作符)
    {
          if(!T->lchild || !T->rchild) return ERROR; //格式错误
          if(T->lchild->data是操作符&&T->lchild->data优先级低于T->data)
          {
              printf("(");
              if(!Print_Expression(T->lchild)) return ERROR;
              printf(")");
           } //注意在什么情况下要加括号
          else if(!Print_Expression(T->lchild)) return ERROR;
          if(T->rchild->data是操作符&&T->rchild->data优先级低于T->data)
          {
              printf("(");
              if(!Print_Expression(T->rchild)) return ERROR;
              printf(")");
          }
          else if(!Print_Expression(T->rchild)) return ERROR;
     }
     else return ERROR; //非法字符
  return OK;
}//Print_Expression
```

40．一棵二叉树的繁茂度定义为各层结点数的最大值与树的高度的乘积。试写一算法，求二叉树的繁茂度。

【分析】关键问题是计算各层结点数的最大值以及树的高度，使用辅助数组。

【解答】算法如下：

```
typedef struct{
                BTNode node;
                int layer;
                int layer;
           } BTNRecord; //包含结点所在层次的记录类型
int FanMao(Bitree T)//求一棵二叉树的"繁茂度"
{
    int count[]; //count数组存放每一层的结点数
    InitQueue(Q); //Q的元素为BTNRecord类型
    EnQueue(Q, {T, 0});
    while(!QueueEmpty(Q))
    {
        DeQueue(Q, r);
        count[r.layer]++;
        if(r.node->lchild) EnQueue(Q, {r.node->lchild, r.layer+1});
        if(r.node->rchild) EnQueue(Q, {r.node->rchild, r.layer+1});
    } //利用层序遍历来统计各层的结点数
    h=r.layer; //最后一个队列元素所在层就是树的高度
    for(maxn=count[0], i=1;count[i];i++)
       if(count[i]>maxn) maxn=count[i]; //求层最大结点数
    return h*maxn;
}//FanMao
```

41．试编写算法，求给定二叉树上从根结点到叶子结点的一条其路径长度等于树的深度减一的路径（即列出从根结点到该叶子结点的结点序列），若这样的路径存在多条，则输出路径终点（叶子结点）在“最左”的一条。

【分析】利用递归算法求得结点深度，再利用递归寻找深度减一的结点。

【解答】算法如下：

```
int maxh;
Status Printpath_MaxdepthS1(Bitree T)//求深度等于树高度减一的最靠左的结点
{
    Bitree pathd;
    maxh=Get_Depth(T);
    if(maxh<2) return ERROR; //无符合条件结点
    Find_h(T, 1);
    return OK;
}//Printpath_MaxdepthS1
void Find_h(Bitree T, int h)//寻找深度为maxh-1的结点
{
    path[h]=T;
    if(h==maxh-1)
    {
        for(i=1;path[i];i++) printf("%c", path[i]->data);
        exit; //打印输出路径
    }
    else
    {
        if(T->lchild) Find_h(T->lchild, h+1);
        if(T->rchild) Find_h(T->rchild, h+1);
    }
    path[h]=NULL; //回溯
}//Find_h
```

42．已知一棵完全二叉树存于顺序表sa中，sa.elem[sa.last]中存放各结点的数据元素。试编写算法由此顺序存储结构建立该二叉树的二叉链表。

【分析】由于此题目给的条件是完全二叉树的顺序存储结构，则可按层序遍历建立二叉链表，注意空表的处理和根指针必须是变参。

【解答】算法如下：

```
Status CreateBitree_SqList(Bitree &T, SqList sa)//根据顺序存储结构建立二叉链表
{
    Bitree ptr[sa.last+1]; //该数组储存与sa中各结点对应的树指针
    if(!sa.last)
    {
        T=NULL; //空树
        return;
    }
    ptr[1]=(BTNode*)malloc(sizeof(BTNode));
    ptr[1]->data=sa.elem[1]; //建立树根
    T=ptr[1];
```

```
    for(i=2;i<=sa.last;i++)
    {
        if(!sa.elem[i]) return ERROR; //顺序错误
        ptr[i]=(BTNode*)malloc(sizeof(BTNode));
        ptr[i]->data=sa.elem[i];
        j=i/2; //找到结点i的双亲j
        if(i-j*2) ptr[j]->rchild=ptr[i]; //i是j的右孩子
        else ptr[j]->lchild=ptr[i]; //i是j的左孩子
    }
    return OK;
}//CreateBitree_SqList
```

43. 为二叉链表的结点增加DescNum域。试写一算法，求二叉树的每个结点的子孙数目并存入其DescNum域。请给出算法的时间复杂度。

【分析】在下面的算法中，为了能用一个统一的公式计算子孙数目，所以当T为空指针时，要返回-1而不是0。该算法时间复杂度为$O(n)$，n为树结点总数。

【解答】算法如下：

```
int DescNum(Bitree T)//求树结点T的子孙总数填入DescNum域中，并返回该数
{
    if(!T) return -1;
    else d=(DescNum(T->lchild)+DescNum(T->rchild)+2); //计算公式
    T->DescNum=d;
    return d;
}//DescNum
```

44. 试写一个算法，在先序后继线索二叉树中，查找给定结点*p在先序序列中的后继（假设二叉树的根结点未知）。并讨论实现此算法对存储结构有何要求？

【解答】算法如下：

```
BTNode *PreOrder_Next(BTNode *p)//在先序后继线索二叉树中查找结点p的先序后继，并返回指针
{
    if(p->lchild) return p->lchild;
    else return p->rchild;
}//PreOrder_Next
```

45. 试写一个算法，在后序后继线索二叉树中，查找给定结点*p在后序序列中的后继（二叉树的根结点指针并未给出）。并讨论实现算法对存储结构有何要求？

【解答】算法如下：

```
Bitree PostOrder_Next(Bitree p)//在后序后继线索二叉树中查找结点p的后序后继，并返回指针
{
    if(p->rtag) return p->rchild; //p有后继线索
    else if(!p->parent) return NULL; //p是根结点
        else if(p==p->parent->rchild) return p->parent; //p是右孩子
           else if(p==p->parent->lchild&&p->parent->tag)
                 return p->parent; //p是左孩子且双亲没有右孩子
            else //p是左孩子且双亲有右孩子
```

```
            {
                q=p->parent->rchild;
                while(!q->ltag || !q->rtag)
                {
                   if(!q->ltag) q=q->lchild;
                   else q=q->rchild;
                }
            } //从p的双亲的右孩子向下走到底
    return q;
}//PostOrder_Next
```

46．试写一个算法，在中序全线索二叉树的结点*p之下，插入一棵以结点*x为根，只有左子树的中序全线索二叉树，使*x为根的二叉树成为*p的左子树。若*p原来有左子树，则令它为*x的右子树。完成插入之后的二叉树应保持全线索化特性。

【解答】算法如下：

```
Status Insert_BiThrTree(BiThrTree &T, BiThrTree &p, BiThrTree &x)//在中序线索二叉树T的结点p
                                                                  //下插入子树x
{
    if(!p->ltag&&!p->rtag) return INFEASIBLE; //无法插入
    if(p->ltag) //x作为p的左子树
    {
        s=p->lchild; //s为p的前驱
        p->ltag=Link;
        p->lchild=x;
        q=x;
        while(q->lchild) q=q->lchild;
        q->lchild=s; //找到子树中的最左结点，并修改其前驱指向s
        q=x;
        while(q->rchild) q=q->rchild;
        q->rchild=p; //找到子树中的最右结点，并修改其前驱指向p
    }
    else //x作为p的右子树
    {
        s=p->rchild; //s为p的后继
        p->rtag=Link;
        p->rchild=x;
        q=x;
        while(q->rchild) q=q->rchild;
        q->rchild=s; //找到子树中的最右结点，并修改其前驱指向s
        q=x;
        while(q->lchild) q=q->lchild;
        q->lchild=p; //找到子树中的最左结点，并修改其前驱指向p
    }
    return OK;
}//Insert_BiThrTree
```

47．编写算法完成下列操作：无重复地输出以孩子兄弟链表存储的树T中所有的边。输

出的形式为（k_1，k_2），…，（k_i，k_j），…，其中k_i和k_j为根结点中的结点标识。

【分析】有一种算法：修改二叉树遍历的递归算法，使其参数表增加参数father，它指向被访问的当前结点在树中的双亲结点。

【解答】算法如下：

```
void Print_CSTree(CSTree T)//输出孩子兄弟链表表示的树T的各边
{
    for(child=T->firstchild;child;child=child->nextsib)
    {
        printf("(%c, %c), ", T->data, child->data);
        Print_CSTree(child);
    }
}//Print_CSTree
```

48. 试编写算法，对一棵以孩子-兄弟链表表示的树统计叶子的个数。

【解答】算法如下：

```
int LeafCount_CSTree(CSTree T)//求孩子兄弟链表表示的树T的叶子数目
{
    if(!T->firstchild) return 1; //叶子结点
    else
    {
        count=0;
        for(child=T->firstchild;child;child=child->nextsib)
        count+=LeafCount_CSTree(child);
        return count; //各子树的叶子数之和
    }
}//LeafCount_CSTree
```

49. 试编写算法，求一棵以孩子-兄弟链表表示的树的度。

【解答】算法如下：

```
int GetDegree_CSTree(CSTree T)//求孩子兄弟链表表示的树T的度
{
    if(!T->firstchild) return 0; //空树
    else
    {
        degree=0;
        for(p=T->firstchild;p;p=p->nextsib) degree++;//本结点的度
        for(p=T->firstchild;p;p=p->nextsib)
        {
            d=GetDegree_CSTree(p);
            if(d>degree) degree=d; //孩子结点的度的最大值
        }
        return degree;
    }//else
}//GetDegree_CSTree
```

50. 对以孩子-兄弟链表表示的树编写计算树的深度的算法。

【解答】算法如下：

```
int GetDepth_CSTree(CSTree T)//求孩子兄弟链表表示的树T的深度
{
    if(!T) return 0; //空树
    else
    {
        for(maxd=0, p=T->firstchild;p;p=p->nextsib)
          if((d=GetDepth_CSTree(p))>maxd) maxd=d; //子树的最大深度
        return maxd+1;
    }
}//GetDepth_CSTree
```

51. 对以孩子链表表示的树编写计算树的深度的算法。

【解答】算法如下：

```
int GetDepth_CTree(CTree A)//求孩子链表表示的树A的深度
{
    return SubDepth(A.r);
}//GetDepth_CTree
int SubDepth(int T)//求子树T的深度
{
    if(!A.nodes[T].firstchild) return 1;
    for(sd=1, p=A.nodes[T].firstchild;p;p=p->next)
      if((d=SubDepth(p->child))>sd) sd=d;
           return sd+1;
}//SubDepth
```

52. 对以双亲表表示的树编写计算树的深度的算法。

【解答】算法如下：

```
int GetDepth_PTree(PTree T)//求双亲表表示的树T的深度
{
    maxdep=0;
    for(i=0;i<T.n;i++)
    {
        dep=0;
        for(j=i;j>=0;j=T.nodes[j].parent) dep++; //求每一个结点的深度
        if(dep>maxdep) maxdep=dep;
    }
    return maxdep;
}//GetDepth_PTree
```

53. 已知一棵二叉树的前序序列和中序序列分别存于两个一维数组中，试编写算法建立该二叉树的二叉链表。

【分析】树具有这样一个性质，即一棵子树在前序和中序序列中所占的位置总是连续的，因此，就可以用起始下标和终止下标来确定一棵子树。Pre-start、Pre-end、In-start和In-end分别指示子树在前序子序列里的起始下标、终止下标和中序子序列里的起始下标

和终止下标。

【解答】算法如下：

```
char Pred, Ind; //假设前序序列和中序序列已经分别储存在数组Pre和In中
Bitree Build_Sub(int Pre_Start, int Pre_End, int In_Start, int In_End)//由子树的前序和中序序列建立其
                                                                    //二叉链表
{
    sroot=(BTNode*)malloc(sizeof(BTNode)); //建根
    sroot->data=Pre[Pre_Start];
    for(i=In_Start;In[i]!=sroot->data;i++); //在中序序列中查找子树根
       leftlen=i-In_Start;
    rightlen=In_End-i; //计算左右子树的大小
    if(leftlen)
    {
        lroot=Build_Sub(Pre_Start+1, Pre_Start+leftlen, In_Start, In_Start+leftlen-1);
        sroot->lchild=lroot;
    } //建左子树，注意参数表的计算
    if(rightlen)
    {
        rroot=Build_Sub(Pre_End-rightlen+1, Pre_End, In_End-rightlen+1, In_End);
        sroot->rchild=rroot;
    } //建右子树，注意参数表的计算
    return sroot; //返回子树根
}//Build_Sub
main()
{
    ...
    Build_Sub(1, n, 1, n); //初始调用参数，n为树结点总数
    ...
}
```

54.假设有n个结点的树T采用了双亲表示法，写出由此建立树的孩子-兄弟链表的算法。

【分析】设一个辅助指针数组p，初始时使每个指针指向一个结点，且置结点数据域的值为双亲表中相应结点的数据元素。以后的工作只是建立这些结点的链域。

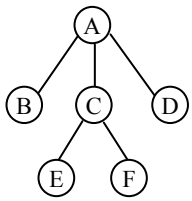
【解答】算法如下：

```
typedef struct{
                CSNode *ptr;
                CSNode *lastchild;
            } NodeMsg; //结点的指针和其最后一个孩子的指针
Status Bulid_CSTree_PTree(PTree T)//由树T的双亲表构造其孩子兄弟链表
{
    NodeMsg Treed;
    for(i=0;i<T.n;i++)
    {
        Tree[i].ptr=(CSNode*)malloc(sizeof(CSNode));
        Tree[i].ptr->data=T.node[i].data; //建结点
        if(T.nodes[i].parent>=0) //不是树根
```

```
        {
            j=T.nodes[i].parent; //本算法要求双亲表必须是按层序存储
            if(!(Tree[j].lastchild)) //双亲当前还没有孩子
                Tree[j].ptr->firstchild=Tree[i].ptr; //成为双亲的第一个孩子
            else //双亲已经有了孩子
               Tree[j].lastchild->nextsib=Tree[i].ptr; //成为双亲最后一个孩子的下一个兄
弟
            Tree[j].lastchild=Tree[i].ptr; //成为双亲的最后一个孩子
        }//if
    }//for
}//Bulid_CSTree_PTree
```

55. 假设以二元组（F，C）的形式输入一棵树的诸边（其中F表示双亲结点的标识，C表示孩子结点标识），且在输入的二元组序列中，C是按层次顺序出现的。F='^'时，C为根结点标识，若C也为'^'，则表示输入结束。例如，如下所示树的输入序列为：

```
^A
AB
AC
AD
CE
CF
^ ^
```



试编写算法，由输入的二元组序列建立树的孩子-兄弟链表。

【分析】熟悉孩子兄弟表示法的结构，同时在对本题的解答中应体会到时间和空间是一对互斥的问题，例如在本题中，牺牲一定的空间设一右child指针，可大大减少检索时间，在平时编程中应体会这种关系。

【解答】算法如下：

```
typedef struct{
                char data;
                CSNode *ptr;
                CSNode *lastchild;
          } NodeInfo; //结点数据，结点指针和最后一个孩子的指针
Status CreateCSTree_Duplet(CSTree &T)//输入二元组建立树的孩子兄弟链表
{
    NodeInfo Treed;
    n=1;k=0;
    if(getchar()!='^') return ERROR; //未按格式输入
    if((c=getchar())=='^') T=NULL; //空树
    Tree[0].ptr=(CSNode*)malloc(sizeof(CSNode));
    Tree[0].data=c;
    Tree[0].ptr->data=c;
    while((p=getchar())!='^'&&(c=getchar())!='^')
    {
        Tree[n].ptr=(CSNode*)malloc(sizeof(CSNode));
```

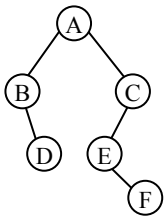
```
        Tree[n].data=c;
        Tree[n].data=c;
        Tree[n].ptr->data=c;
        for(k=0;Tree[k].data!=p;k++); //查找当前边的双亲结点
           if(Tree[k].data!=p) return ERROR; //未找到:未按层序输入
        r=Tree[k].ptr;
        if(!r->firstchild)
            r->firstchild=Tree[n].ptr;
        else Tree[k].lastchild->nextsib=Tree[n].ptr;
           Tree[k].lastchild=Tree[n].ptr; //这一段含义同上一题
        n++;
    }//while
    return OK;
}//CreateCSTree_Duplet
```

56．已知一棵树的由根至叶子结点按层次输入的结点序列及每个结点的度（每层中自左到右输入），试写出构造此树的孩子-兄弟链表的算法。

【解答】算法如下：

```
Status CreateCSTree_Degree(char node[ ], int degree[ ])//由结点的层序序列和各结点的度构造树的孩子
                                                    //兄弟链表
{
    CSNode * ptrd; //树结点指针的辅助存储
    ptr[0]=(CSNode*)malloc(sizeof(CSNode));
    i=0;k=1; //i为当前结点序号，k为当前孩子的序号
    while(node[i])
    {
        ptr[i]->data=node[i];
        d=degree[i];
        if(d)
        {
            ptr[k++]=(CSNode*)malloc(sizeof(CSNode)); //k为当前孩子的序号
            ptr[i]->firstchild=ptr[k]; //建立i与第一个孩子k之间的联系
            for(j=2;j<=d;j++)
            {
                  ptr[k++]=(CSNode*)malloc(sizeof(CSNode));
                  ptr[k-1]->nextsib=ptr[k]; //当结点的度大于1时，为其孩子建立兄弟链表
            }//for
        }//if
        i++;
    }//while
}//CreateCSTree_Degree
```

57．假设以二叉链表存储的二叉树中，每个结点所含数据元素均为单字母，试编写算法，按树状打印二叉树的算法。例如，左下二叉树印为右下形状。



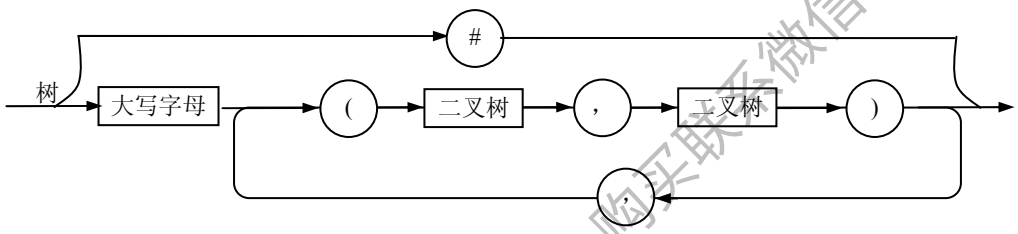
	C		
			F
		E	
A			
		D	
	B		

【分析】采用带层次信息的中序遍历思想，按题目要求，顺序为先右后左。

【解答】算法如下：

```
void Print_BiTree(BiTree T, int i)//按树状打印输出二叉树的元素，i表示结点所在层次，初次调用时i=0
{
    if(T->rchild) Print_BiTree(T->rchild, i+1);
    for(j=1;j<=i;j++) printf(" "); //打印i个空格以表示出层次
    printf("%c\n", T->data); //打印T元素，换行
    if(T->lchild) Print_BiTree(T->rchild, i+1);
}//Print_BiTree
```

58. 如果用大写字母标识二叉树结点，则一棵二叉树可以用符合下面语法图的字符序列表示。试写一个递归算法，由这种形式的字符序列，建立相应的二叉树的二叉链表存储结构。



例如：57题所示二叉树的输入形式为A（B（#，D），C（E（#，F），#））。

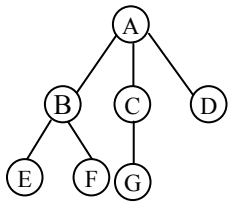
【分析】不妨称这种字符序列为“二叉广义表”。自顶向下的识别策略通常导致结构良好的算法。

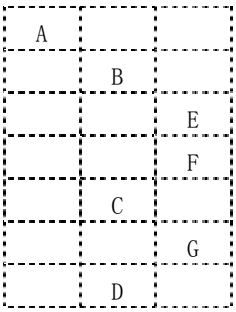
【解答】算法如下：

```
Status CreateBiTree_GList(BiTree &T)//由广义表形式的输入建立二叉链表
{
    c=getchar();
    if(c=='#') T=NULL; //空子树
    else
    {
        T=(CSNode*)malloc(sizeof(CSNode));
        T->data=c;
        if(getchar()!='(') return ERROR;
        if(!CreateBiTree_GList(pl)) return ERROR;
        T->lchild=pl;
```

```
        if(getchar()!=', ') return ERROR;
        if(!CreateBiTree_GList(pr)) return ERROR;
        T->rchild=pr;
        if(getchar()!=')') return ERROR; //这些语句是为了保证输入符合A(B, C)的格式
    }
    return OK;
}//CreateBiTree_GList
```

59. 假设树上每个结点所含的数据元素为一个字母，并且以孩子-兄弟链表为树的存储结构，试写一个按凹入表方式打印一棵树的算法。例如，左下树印为右下形状。





【分析】利用递归思想容易解决此题。

【解答】算法如下：

```
void Print_CSTree(CSTree T, int i)//按凹入表形式打印输出树的元素，i表示结点所在层次，初次调用
                                  //时i=0
{
    for(j=1;j<=i;j++) printf(" "); //留出i个空格以表现出层次
    printf("%c\n", T->data); //打印元素，换行
    for(p=T->firstchild;p;p=p->nextsib)
      Print_CSTree(p, i+1); //打印子树
}//Print_CSTree
```

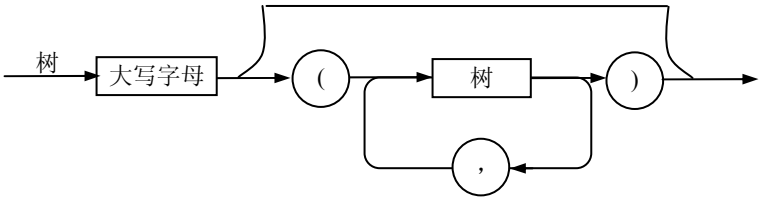
60. 以孩子链表为树的存储结构，重做59题。

【解答】算法如下：

```
void Print_CTree(int e, int i)//按凹入表形式打印输出树的元素，i表示结点所在层次
{
    for(j=1;j<=i;j++) printf(" "); //留出i个空格以表现出层次
    printf("%c\n", T.nodes[e].data); //打印元素，换行
    for(p=T.nodes[e].firstchild;p;p=p->next)
        Print_CSTree(p->child, i+1); //打印子树
}//Print_CSTree
main()
{
```

```
    ...
    Print_CTree(T.r, 0); //初次调用时i=0
    ...
}//main
```

61. 若用大写字母标识树的结点，则可用带标号的广义表形式表示一棵树，其语法图如下所示：



例如，59题中的树可用下列形式的广义表表示：

A(*B*(*E*, *F*), *C*(*G*), *D*)

试写一递归算法，由这种广义表表示的字符序列构造树的孩子-兄弟链表（提示：按照森林和树相互递归的定义写两个互相递归调用的算法，语法图中一对圆括号内的部分可看成为森林的语法图）。

【分析】根据语法图，在此只讨论非空森林，且假设输入的字符序列符合语法图，即不讨论输入出错的情况。建森林和建树的间接递归算法分别如下所示。

【解答】算法如下：

```
Status CreateF ( CSTree &FS )
{
        //本算法识别一个待输入的带标号的广义表形式的字符序列，建立森林的
        //孩子-兄弟链表存储结构，全局量ch中含当前读入的字符，FS为指向根
        //结点的指针。算法结束时，ch内含森林广义表串之后的字符。
        if (CreateT(FS)= = OVERFLOW ) return ERROR;
                          //建森林中第一棵树
        p=FS;
        while (ch = = ', ' ) {   //建森林中其余各棵树
             CreateT (p->nextsibling);
             P=p->nextsibling;
        }
        p->nextsibling=NULL;
        return OK;
} //CreateF
Status CreateT (CSTree &T ) {
     //本算法识别待输入的树广义表串，建立树的孩子-兄弟链表，T指向根结点。
     //算法结束时，ch内含该串闭括号之后的字符。
     scanf (ch);
     if (!(T= (CSTree ) malloc (sizeof (CSNode)))) return OVERFLOW;
     T->data=ch;              //建立根结点
     scanf (ch);               //识别大写字母之后的一个字符
     if (ch! = ' (') T->firstchild=NULL;        //叶子结点
```

```
    else {
         CreateF (T->firstchild);   //建子树森林
         scanf (ch);              //读取该树广义表串的闭括号
         }
         return OK;
} //CreateT
```

62．试写一递归算法，由61题定义的广义表表示法的字符序列，构造树的孩子链表。

【分析】在下面的算法中，POS变量起着“分配”结点在表中的位置的作用，是按先序序列从上向下分配，因此树根T.r一定等于0，而最终的POS值就是结点数T.n。

【解答】算法如下：

```
char c;
int pos=0; //pos是全局变量，指示已经分配到了哪个结点
Status CreateCTree_GList(CTree &T, int &i)//由广义表形式的输入建立孩子链表
{
    c=getchar();
    T.nodes[pos].data=c;
    i=pos++; //i是局部变量，指示当前正在处理的子树根
    if((c=getchar())=='(') //非叶子结点
    {
        CreateCTree_GList();
        p=(CTBox*)malloc(sizeof(CTBox));
        T.nodes[i].firstchild=p;
        p->child=pos; //建立孩子链的头
        for(;c==',';p=p->next) //建立孩子链
        {
             CreateCTree_GList(T, j); //用j返回分配得到的子树根位置
             p->child=j;
             p->next=(CTBox*)malloc(sizeof(CTBox));
        }
        p->next=NULL;
        if((c=getchar())!=')') return ERROR; //括号不配对
    }//if
    else T.nodes[i].firstchild=NULL; //叶子结点
    return OK;
}//CreateBiTree_GList
```

63．试写一递归算法，以61题给定的树的广义表表示法的字符序列形式输出以孩子链表表示的树。

【解答】算法如下：

```
void PrintGList_CTree(CTree T, int i)//按广义表形式输出孩子链表表示的树
{
    printf("%c", T.nodes[i].data);
    if(T.nodes[i].firstchild) //非叶子结点
    {
        printf("(");
        for(p=T->firstchild;p;p=p->nextsib)
```

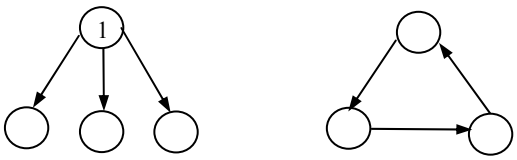
```
        {
           PrintGlist_CSTree(T, p->child);
           if(p->nextsib) printf(", "); //最后一个孩子后面不需要加逗号
        }
        printf(")");
    }//if
}//PrintGlist_CTree
```

6.4 考研真题分析

例题6-1 （中国科学院计算技术研究所1999年试题）

一棵非空的有向树中恰有一个顶点入度为0，其他顶点入度为1。但恰有一个顶点入度为0，其他顶点入度为1的有向图却不一定是一棵有向树。请举例说明之。

【解答】如下图所示的有向图，除顶点1入度为0外，其他每个顶点的入度都为1，但此图却不是有向树。



即一个有向树外加一个有向环所构成的有向图都不是一棵有向树。

例题6-2 （中国科学院软件研究所1998年试题）

若一个具有N个顶点，K条边的无向图是一个森林（$N>K$），则该森林中必有_____棵树。

A. K　　B. N　　C. $N-K$　　D. 1

【解答】设此森林中有m棵树，每棵树具有的顶点数为v_i($1\leqslant i\leqslant m$)，则

$v_1+v_2+\cdots+v_m=N$,　　①

$(v_1-1)+(v_2-1)+\cdots+(v_m-1)=K$　　②

①-②得：

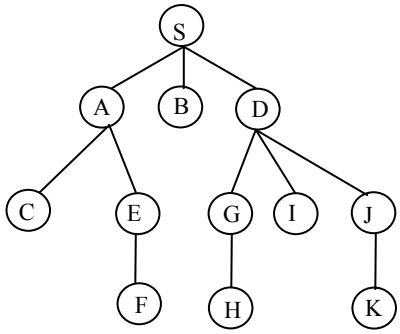
$m=N-K$

答案为C。

例题6-3 （清华大学1998年试题）

假设先根次序遍历某棵树的结点次序为SACEFBDGHIJK，后根次序遍历该树的结点次序为CFEABHGIKJDS，要求画出这棵树。

【解答】



例题6-4　（中国科学院计算技术研究所1999年试题）

深度为K（设根的层数为1）的完全二叉树至少有_____个结点，至多有_____个结点，k和结点数n之间的关系为_____。

【分析】当第k层只有1个结点时，深度为k的完全二叉树所具有结点数最少，为$2^0+2^1+\cdots+2^{k-2}+1=2^{k-1}$；当第$k$层有$2^{k-1}$个结点时，深度为$k$的完全二叉树所具有的结点数最多，为$2^0+2^1+\cdots+2^{k-1}=2^k-1$。

【解答】2^{k-1}，2^k-1，$k=1+\lfloor\log_2 n\rfloor$或$k=\lceil\log_2(n+1)\rceil$。

例题6-5　（中国科学院软件研究所1998年试题）

若从二叉树的任一结点出发到根的路径上所经过的结点序列按其关键字有序，则该二叉树是_____。

A. 二叉排序树　　B. 哈夫曼树　　C. 堆

【分析】哈夫曼树与其结点的关键字无关；一般情况下，从二叉排序树任一结点出发到根的路径上所经过的结点序列不是有序的；而在堆中，不管是左儿子还是右儿子，其关键字与其父结点的关键字的大小是一定的。

【解答】C

例题6-6　（中国科学院计算技术研究所1999年试题）

_____的遍历仍需要栈的支持。

A. 前序线索树　　B. 中序线索树　　C. 后序线索树

【解答】C

例题6-7　（中国科学技术大学1999年试题）

对于前序遍历和中序遍历结果相同的二叉树为_____；对于前序遍历和后序遍历结果相同的二叉树为_____。

A. 一般二叉树　　B. 只有根结点的二叉树

C. 根结点无左孩子的二叉树　　D. 根结点无右孩子的二叉树
E. 所有的结点只有左子树的二叉树　　F. 所有结点只有右子树的二叉树

【解答】F，B

例题6-8　（中国科学院软件研究所1999年试题）

判断正误：
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。

【分析】二叉排序树（Binary Sort Tree）或者是一棵空树，或者是具有下列性质的二叉树：（1）若它的左子树不空，则左子树上所有结点的值均小于它的根结点的值；（2）若它的右子树不空，则它的右子树上所有结点的值均大于它的根结点的值；（3）它的左、右子树也分别为二叉排序树。

【解答】正确。

例题6-9　（中国科学院计算技术研究所1999年试题）

若度为m的哈夫曼树中，其叶子结点个数为n，则非叶子结点的个数为_____。

A. $n-1$　B. $\left\lfloor \frac{n}{m} \right\rfloor -1$　C. $\left\lceil \frac{n-1}{m-1} \right\rceil$　D. $\left\lceil \frac{n}{m-1} \right\rceil -1$　E. $\left\lceil \frac{n+1}{m+1} \right\rceil -1$

【分析】在构造度为m的哈夫曼树的过程中，每次把m个子结点合并为一个父结点（第一次可能合并少于m个子结点），每次合并减少$m-1$个结点，从n个叶子结点减少到最后一个父结点，共需$\left\lceil \frac{n-1}{m-1} \right\rceil$次合并，每次合并增加一个非叶子结点。

【解答】C

例题6-10　（清华大学1998年试题）

① 设T是具有n个内结点的扩充二叉树，I是它的内路径长度，E是它的外路径长度，试利用归纳法证明$E=I+2n$，$n\geqslant 0$。

② 利用①的结果，试说明成功查找的平均比较次数s与不成功查找的平均比较次数u之间的关系，可用公式

$$s=(1+\frac{1}{n})u-1\quad n\geq 1$$

表示。

【解答】

①【证明】$n=1$成立

设$n=k$成立，$E_k=I_k+2k$

当$n=k+1$时，设增加结点的深度为D_k

$I_{k+1}=I_k+D_k$

$E_{k+1}=E_k-D_k+2(D_k+1)$

$= E_k+D_k+2=I_k+2k+ D_k+2$

$=I_{k+1}+2(k+1) =I_{k+1}+2n$

所以当$n=k+1$时也成立。

结论成立。

② 根据二叉树性质：度为0的结点数n_0=度为2的结点数n_2+1

所以，外结点数=n+1

s=(查找成功总的比较次数) /(内结点数)=I/n　　（1）

u=(查找失败总的比较次数)/(外结点数)

=$(E-n)/(n+1)=(I+n)/(n+1)$　　（2）

由（1）、（2）可得　　$I=ns=(n+1)u-n$

$$s=(1+\frac{1}{n})u-1$$

例题6-11　（中国科学院自动化研究所1999年试题）

试写一个所给定二叉树是否为二叉排序树的算法，设此二叉树以二叉链表为存储结构，且树中结点的关键字均不同。

【分析】二叉排序树（Binary Sort Tree）定义如下，二叉排序树或者是一棵空树；或者是具有下列性质的树。

（1）若它的左子树不空，则左子树上所有结点的值均小于它的根结点的值；

（2）若它的右子树不空，则右子树上所有结点的值均小于它的根结点的值；

（3）它的左右子树也分别为二叉排序树。

根据它的递归定义，给出递归的解法。

【解答】算法如下：

```
typedef struct node{
        struct node    *left, *right;
        DataType       data;
}NODE, *TREE;

int IsBST0(TREE t, int *pmax, int *pmin)
{
    int max, min;
    if(!t->left)   *pmin=t->data;
    else{
        if(!IsBST0(t->left, &max, &min))
            return 0;
        if(max>t->data)   return 0;
        *pmin=min;
    }
    if(!t->right)  *pmax=t->data;
    else{
        if(!IsBST0(t->right, &max, &min))
            return 0;
        if(min>t->data)   return 0;
```

```
            *pmin=max;
        }
        return-1;
    }
    int  IsBST(TREE t){
        int max, min;
        if(!t) return -1;
        else return IsBST0(t, &max, &min);
    }
```

例题6-12　（中国科学院自动化研究所1998年试题）

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。试问该树中有多少个叶子结点？

【解答】设该树中叶子结点个数为N_0。

则该树结点个数为$N_0+N_1+\cdots+N_m=\sum_{i=0}^{m}N_i$

又该树结点个数为$1+\sum_{i=1}^{m}iN_i$

所以 $N_0+\sum_{i=1}^{m}N_i=1+\sum_{i=1}^{m}iN_i$

$$N_0=1+\sum_{i=1}^{m}(i-1)N_i$$

例题6-13　（东北大学1999年试题）

试用C语言编写一个递归函数，函数的输入为一个带括号的包含加、减、乘、除四则运算表达式的字符串，例如“(a+b)*(c+d)”，该函数根据运算符优先级来构造一棵二叉树，树的根结点为优先级最低的运算符。该函数最终返回二叉树的根结点。

【分析】可以按中序或后序递归生成表达式树。关键在于对优先级的判定，可以首先对表达式进行分析，去掉括号，并求出各个操作符的优先级。

【解答】递归函数如下：

```
typedef struct node{
    struct node     *left, *right;
    char op;
}NODE, *TREE;

char op[255];       //存放经过分析后，去掉括号的表达式
int opw[255];       //存放各个操作符的优先级
                    //+-为1；×÷为2；括号内加2
int len;            //op表达式长度
char *exp;          //初始表达式
TREE expt;          //表达式树
```

```
void analysis(char* exp){
     //分析表达式，去掉括号，并求各个操作符的优先级
     int  w=0;       //括号的权重
     char *p=exp;
     while(*p){
          switch(*p){
               case  '(':     w+=2;
                              break;
               case  ')':          w-=2;
                              break;
               case '+':
               case '-':           op[len]=*p;
                              opw[len]=w+1;
                              len++;


                              break;

               case '*':
               case '/':           op[len]=*p;
                              opw[len]=w+2;
                              len++;
                              break;
               default:
                              op[len]=*p;
                              opw[len]=0;
                              len++;
          }//end of switch
          p++;
     }//end of while
}//end of analysis

int findop(int l, int h){
     //在opw[l..h]中找出优先级最高的元素
     int max w, k, i;
     maxw=opw[l];
     k=l;
     for(i=l+1;i<=h;i++)
          if(max<opw[i]){
               max=opw[i];
               k=i;
          }
     return k;
}

TREE createexp(int l, int h){
     //为表达式op[l..h]建立表达式树
     TREE  t;
     int k;
     t=(TREE)malloc(sizeof(NODE));
```

```
        if(l==h){
            t->left=t->right=NULL;
            return t;
        }
        else{
            k=findop(l, h);
            t->left=createexp(l, k-1);
            t->right=createexp(k+1, h);
            t->op=op[k];
        }
        return t;
    }

    main(){


        analysis(exp);
        expt=createexp(0, len-1);
    }
```

例题6-14　（清华大学1999年试题）

一棵高度为h的满k叉树有如下性质：根结点所在的层次为0；第h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，如果按层次自顶向下，同一层自左向右，顺序从1开始对全部结点进行编号，试问：

① 各层的结点个数是多少？

② 编号为i的结点的双亲结点（若存在）的编号是多少？

③ 编号为i的结点的第m个孩子结点（若存在）的编号是多少？

④ 编号为i的结点有右兄弟的条件是什么？其右兄弟结点的编号是多少？

【解答】① 第i层的结点个数为k^i。

② 编号为i的结点的双亲结点的编号是$\left\lfloor \frac{i+k-2}{k} \right\rfloor$。

③ 编号为i的结点的第m个孩子结点的编号是$k(i-1)+m+1$。

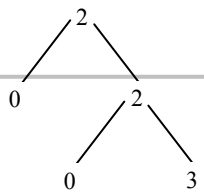
④ 编号为i的结点有兄弟的条件是：i除以k的余数不等于1。其右兄弟结点的编号是$i+1$。

例题6-15　（北京邮电大学1998年试题）

判断正误：

二叉树中除叶子结点外，任一结点x，其左子树根结点的值小于该结点（x）的值；其右子树根结点的值大于等于该结点（x）的值，则此二叉树是二叉排序树。

【分析】二叉排序树的定义是一个递归定义，要求左右子树都是二叉排序树。该命题并不符合定义。可以举出反例，如下图所示。



这棵树满足条件，但不是二叉排序树。

【解答】错误。

例题6-16　（清华大学1999年试题）

二项式展开式的系数为

$C(n, 0)=1$，$C(n, n)=1$，对于$n\geqslant 0$

$C(n, k)=C(n-1, k)+C(n-1, k-1)$，对于$0<k<n$

形成著名的杨辉三角形，如下所示。

					1								n=0
				1		1							n=1
			1		2		1						n=2
		1		3		3		1					n=3
	1		4		6		4		1				n=4
1		5		10		10		5		1			n=5
1		6		15		20		15		6		1	n=6

（1）试写一个递归算法，根据以上公式生成$C(n, k)$。

（2）试画出计算$C(6, 4)$的递归树。

（3）试写一个非递归算法，既不用数组也不用栈，对于任意的$0\leqslant k\leqslant n$计算$C(n, k)$。

【分析】$C(n, k)$的递归定义如下：

$C(n, k)=1$　　$k=0$

$C(n, k)=1$　　$k=n$

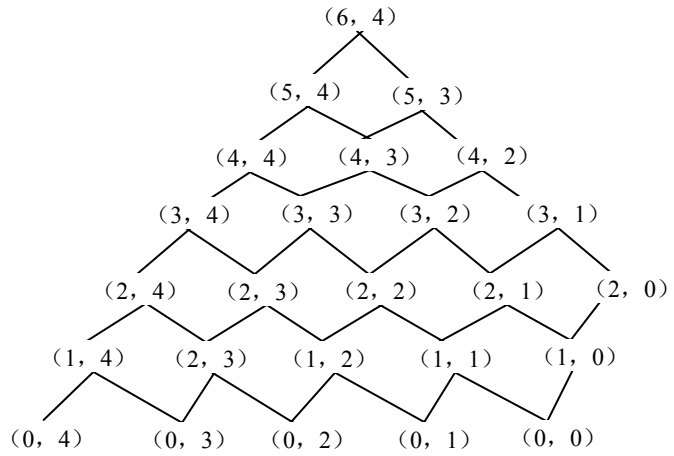
$C(n, k)=C(n-1, k)+C(n-1, k-1)$　　否则

根据二项式定理，$C(n, k)$就等于组合公式$C_n^k=\dfrac{n(n-1)\Lambda\ (n-k+1)}{1*2*\Lambda\ (k-1)k}$。

【解答】（1）$C(n, k)$的递归算法如下：

```
int C(int n, int k){
    if(k==0)return 1;
    if(k==n)return 1;
    return C(n-1, k)+ C(n-1, k);
}
```

（2）$C(6, 4)$的递归树如下所示。



（3）$C(n, k)$的非递归算法如下：

```
int C(int n, int k){
     long  x, y;
     int  i;
     x=y=1;
     for(i=1;i<=k;i++)x*=i;
     for(i=n;i>=n-k+1;i--)y*=i;
     return y/x;
}
```

例题6-17　（北京邮电大学1998年试题）

有n个结点的二叉树，已知叶子结点个数为n_0，写出求度为1的结点的个数n_1的计算公式；若此树是深度为k的完全二叉树，写出n为最小的公式；若二叉树中仅有度为0和度为2的结点，写出求该二叉树结点个数n的公式。

【解答】（1）记度为2的结点个数为n_2

则$n=n_0+n_1+n_2$　　①

又因为除了根结点以外，其余结点均由父结点射出，所以

$n=1+n_1+2*n_2$　②

①②两式得$n_1=n+1-2n_0$

（2）当树是深度为k的完全二叉树时，n的最小值$n_{min}=2^{k-1}$

（3）当二叉树中仅有度为0和度为2的结点时，

$n=n_0+n_2$　　$n=2n_2+1$

所以　　$n_0=n_2+1$

$n=2n_0-1$

例题6-18　（清华大学1999年试题）

下表所示的数据表给出了在一篇有19 710个词的英文行中出现最普遍的15个词的出现频度。

（1）假设一个英文字符等价于$\log_2 26=4.7010$ bits，那么这些词按bits计的平均长度是多少？

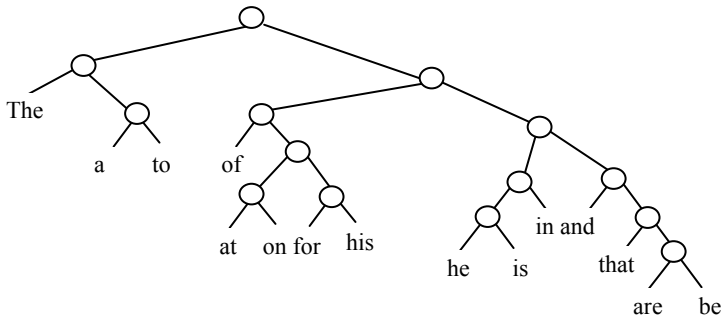
（2）假定一篇正文仅由上述的数据表中的词组成，那么它们的最佳编码是什么？平均长度是多少？

词	The	of	a	to	and	in	that	he	is	at	on	for	his	are	be
出现频度	1192	677	541	518	462	450	242	195	190	181	174	157	138	124	123

【解答】

（1）平均长度 $L=(\log_2 26\sum_{i=1}^{15} f_i l_i)/(\sum_{i=1}^{15} f_i)=2.376*\log_2 26=11.168$。

（2）对应的哈夫曼树如下图所示。



最佳编码如下表所示。

The:00	of:100	a:010	to:011	and:1110	in:1101
that:11110	he:11000	is:11001	at:10100	on:10101	for:10110
His:10111	are:111110	be:111111			

平均长度：$\Sigma_i W_i=3.562$。

例题6-19（北京邮电大学1998年试题）

具有n个结点的完全二叉树，已经顺序存储在一维数组A[1..n]中，下面的算法是将A中顺序存储变为二叉链表存储的完全二叉树。请填入适当语句在下面的空格内，完成上述算法。

```
type AR=array[1..n]of datatype;
     pointer=record;
          data:datatype;
          lchild, rchild:pointer;
     end;
procedure btree(var A:AR;var p:pointer);
     var j:integer;
     procedure createtree(var t:pointer;i:integer)
```

```
    begin
    ____(1)____;
    t^.data:=A[i];
    if____(2)____
        then createtree(____(3)____)
        else t^.lchild:=NIL;
    if____(4)____
        then createtree(____(5)____)
        else t^.lchild:=NIL;
    end;
begin
    j:=____(6)____;
    createtree(p, j);
end;
```

【解答】

(1) new(t)
(2) i*2<=n
(3) t^.lchild, i*2
(4) i*2+1<=n
(5) t^.rchild, i*2+1
(6) 1

例题6-20 （清华大学1999年试题）

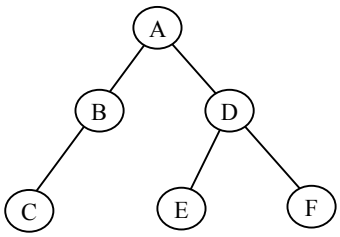
八皇后的问题的最大递归深度是多少？

【解答】八皇后问题的最大递归深度是8。

例题6-21 （北京工业大学1998年试题）

已知二叉树BT各结点的先序、中序遍历列分别为A、B、C、D、E、F和C、B、A、E、D、F，试画出该二叉树。

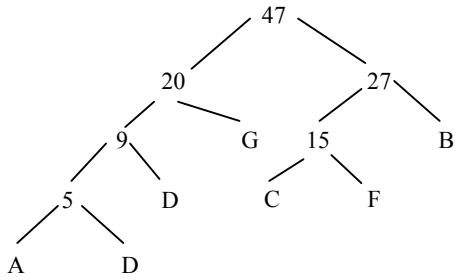
【解答】二叉树如下图所示。



例题6-22 （北京工业大学1998年试题）

已知下列字符A、B、C、D、E、F、G的权值分别为3、12、7、4、2、8、11，试填写出其对应哈夫曼树HT的存储结构的初态和终态。

【解答】对应的哈夫曼树如下图所示。



HT存储结构的初态如下表所示。

Weight	lch	rch	parent
3	0	0	0
12	0	0	0
7	0	0	0
4	0	0	0
2	0	0	0
8	0	0	0
11	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

HT存储结构的修整如下表所示。

Weight	lch	rch	parent
3	0	0	8
12	0	0	12
7	0	0	10
4	0	0	9
2	0	0	8
8	0	0	10
11	0	0	11
5	1	5	9

9	8	4	11
15	3	6	12
20	9	7	13
27	10	2	13
47	11	12	0

例题6-23 （北京工业大学1998年试题）

设某二叉树结点结构为：

```
type bitreptr=↑bnodetp;
     bnodetp=record
          data:integer;
          lchild, rchild:bitreptr
     end;
```

试编写算法，计算每层中结点data域数值大于50的结点个数，并输出这些结点的data域的数值和序号。

【解答】算法如下：

```
type resultarray[1..100]of integer;
procedure level_trav(var bt:bitreptr);  //按层遍历，给结点编号
begin
  n:=0;
  INIT(Q);
  IN(Q, bt);
  while NOT empty(a) do
          begin
               t:=OUT(Q);
               n:=n+1;
               t↑.num:=n;
               IF t↑.lchild<>NIL THEN
                    IN(Q, t↑.lchild);
               IF t↑.lchild<>NIL THEN
                    IN(Q, t↑.rchild);
          END;
END;
PROCEDURE total(bt:bitreptr);
BEGIN
          IF bt<>NIL THEN
               BEGIN
                    h:=h+1;
                    m:=max(m, h);
                    IF bt↑.data>50 THEN
                      BEGIN
                         a[h]:=a[h]+1;
                         writeln(bt↑.data, bt↑.num);
                              //输出结点值与序号
                      END;
```

```
                total(bt↑.lch);
                total(bt↑.rch);
                h:=h-1;
        END;
END;
PROCEDURE main(bt:bitreptr);
BEGIN
   VAR a:resultarray;m:INTEGER;
   Create(bt);
   level_trav(bt);
   m:=0;
   h:=0;
   total(bt);
   FOR i:=1 TO m DO
          writeln(I, a[i]);
END.
```

例题6-24　（清华大学2002年试题）

判断正误：

（1）二叉搜索树可以定义为二叉树的派生类。这样，它可以继承二叉树的所有特性。

（2）将一棵用顺序方式存储的完全二叉树T[n]转换成用二叉链表存储的二叉树，可以编写一个非递归的转换算法，利用栈将T[n]中的结点数据顺序读取出来以建立二叉链表表示的二叉树。

【解答】（1）错误　（2）正确

例题6-25　（清华大学2002年试题）

应用Prim算法求解连通网络的最小生成树问题。

（1）针对右图所示的连通网络。试按如下格式给出在构造最小生成树过程中顺序选出的各条边。

（始顶点号，终顶点号，权值）

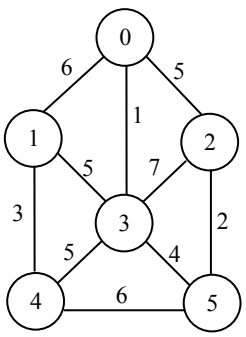
（　　，　　，　　）

（　　，　　，　　）

（　　，　　，　　）

（　　，　　，　　）

（　　，　　，　　）



（2）下面是Prim算法的实现，中间有5个地方缺失，请阅读程序后将它们补上。

```
const int MaxInt=INT_MAX;    //INT_MAX的值在<limits.h>中
const int n=6;//图的顶点数，应由用户定义
typedef int AdjMatrix[n][n];//用二维数组作为邻接矩阵表示
typedef struct{    //生成树的边结点
    int from Vex，toVex;    //边的起点与终点
    int weight;    //边上的权值
}TreeEdgeNode;
```

```
typedef TreeEdgeNode MST[n-1];   //最小生成权定义

void PrimMST(AdjMatrix G, MST T, int rt){
//从顶点rt出发构造图G的最小生成树T，rt成为树的根结点
TreeEdgeNode e;  int i, k=0, min, minpos, v;
for(i=0;i<n;i++)   //初始化最小生成树T
  if(i!=rt){
    T[k].from Vex=rt;
    ①          ;
    T[k++].weight=G[rt][i];
   }
 for(k=0;k<n-1;k++){  //依次求MST的候选边
   ②           ;
   for(i=k;i<n-1;i++)   //遍历当前候选边集合
    if(T[i].weight<min) //选具有最小权值的候选边
    {min=T[i].weight;  ③   ;}
  if(min==MaxInt)  //图不连通，出错处理
     {corr<<"Graph is disconnected! "<<endl;  ④   ;}
   e=T[minpos];  T[minpos]=T[k];   T[k]=e;
   v=T[k].toVex;
   for(i=k+1;i<n-1;i++) //修改候选边集合
      if(G[v][T[i].toVex]<T[i].weight){
        T[i].weight=G[v][T[i].toVex];
            ⑤        ;
      }
   }
}
```

【解答】

（1）

始顶点号	终顶点号	权值
0	3	1
3	5	4
5	2	2
3	4	5
4	1	3

（2）① T[k].tovex=i
② min=MaxInt
③ minpos=i
④ return
⑤ T[i].formVex=v

例题6-26 （中国科学院计算机网络信息中心2001年试题）

单选题：

设根的层数为0，在高度为h的严格二叉树（即无1度结点的二叉树）中，结点总数n满足_____。

A．$2h+1 \leqslant n \leqslant 2^h-1$　　B．$2h-1 \leqslant n \leqslant 2^h-1$

C．$2h-1 \leqslant n \leqslant 2^{h+1}-1$　　D．$2h+1 \leqslant n \leqslant 2^{h+1}-1$

【解答】D

例题6-27　（中国科学院计算机网络信息中心2001年试题）

单选题：

用给定的哈夫曼编码来压缩数据文件，其压缩效率主要取决于_____。

A．文件长度　　B．平均码长　　C．被压缩文件的特征

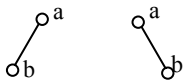
【解答】C

例题6-28　（中国科学院计算机网络信息中心2001年试题）

问答题：

给定二叉树的先序序列和中序序列，能否重构出该二叉树？给定二叉树的先序序列和后序序列呢？若能，试证明之，否则给出反例。

【解答】由先序序列和中序序列能惟一确定一棵二叉树，但先序和后序序列不能惟一确定一棵二叉树，反例为：



先序序列ab，后序序列ba可对应两棵二叉树。

例题6-29　（中国科学院计算机网络信息中心2000年试题）

下述编码中哪一个不是前缀码_____。

A．{00，01，10，11}　　B．{01，00，11}

C．{0，10，110，111}　　D．{1，01，000，001}

【解答】B

例题6-30　（中国科学院计算机网络信息中心2000年试题）

由二叉树的前序遍历和中序遍历序列能确定惟一的一棵二叉树，下面程序的作用是实现由已知某二叉树的前序遍历和中序遍历序列，生成一棵用二叉链表表示的二叉树并打印出后序遍历序列，请写出程序中所缺的语句。

```
#define MAX 100
typedef struct Node{
   char    info;
   struct Node *llink, *rlink;
}TNODE;
char pred[MAX], inod[MAX];

main(int argc, int **argv)
{
```

```
    TNODE   *root;
    if(argc<3)exit 0;
    strcpy(pred, argb[1]);
    strcpy(inod, argv[2]);
    root=restore(pred, inod, strlen(pred));
    postrder(root);
}
TNODE *restore(char *ppos, char *ipos, int n)
{
    TNODE  *ptr;
    char     *rpos;
    int        k;
    if(n<=0)return NULL;
    ptr->info= ____①____;
    for(____②____ ;rpos<ipos+n;rpos++)
        if(*rpos== *ppos)break;


    k= ____③____;
    ptr->llink=restore(ppos+1, ____④____, k);
    ptr->rlink=restore(____⑤____+k, rpos+1, n-1-k);
    return ptr;
}
postorder(TNODE *ptr)
{
   if(ptr==NULL)return;
   postorder(ptr->llink);
   postorder(ptr->rlink);
   printf("%c", ptr->info);
}
```

【解答】

①*ppos或ppos[0]　②rpos=ipos　③rpos-ipos　④ipos或rpos-k　⑤ppos+1

例题6-31 （中国科学院计算机网络信息中心2000年试题）

试求有n个叶子结点的非满的完全二叉树的高度。

【解答】

$$\lfloor \log_2 2n \rfloor \text{or} \lfloor \log_2 n \rfloor + 2 \text{or} \lceil \log_2 (n+1) \rceil + 1$$

例题6-32 （华北计算技术研究所2001年试题）

将一棵树转化为二叉树后，根结点_____左子树。

A. 有　　B. 没有

【解答】A

例题6-33 （华北计算技术研究所2001年试题）

哈夫曼树是带树路径长度最短的树，路径上长度_____结点离根____。

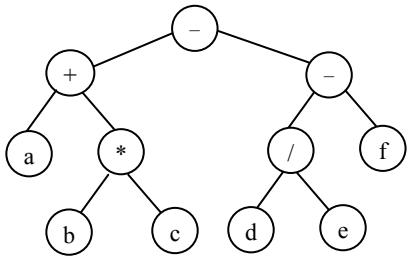
A. 较大　　B. 较小　　C. 较近　　D. 较远

【解答】B，D

例题6-34 （华北计算技术研究所2001年试题）

已知二叉树的先序遍历序列为“-、+、a、*、b、c、-、/、d、e、f”，中序遍历序列为“a、+、b、*、c、-、d、/、e、-、f”，画出此二叉树，并写出它的后序遍历序列。

【解答】二叉树如下所示：



后序遍历序列为“a、b、c、*、+、d、e、/、f、-、-”。

例题6-35 （北京大学2000年试题）

一个二叉树的中序序列和后序序列分别是DCBAEFG和DCBGFEA，请给出该二叉树的先序序列。

【解答】ABCDEFG

例题6-36 （北京大学2000年试题）

试构造按宽方向周游二叉树对应的树林（即依次访问树林的0层，1层，2层……的结点）的算法。

【解答】算法如下：

```
#define  MAXN 50
typedef struct node{int data;
                    struct node*fch *nsib;  /*第一个孩子结点与下一个兄弟结点的指针*/
                  }NODE;
typedef  struct queuetype{int qa, qe; /*队列首尾指针*/
                          NODE *item[MAXN];  /*队列数组*/
                         }QUEUE;
void  whidthtrave(forest C NODE *ht)
{ QUEUE  higherq,  lowerq; /*存放高层元素和低层元素的队列*/
   NODE  *p;
   int i;
   if(ht == NULL) return;  /*空树直接返回*/
   higherq.qa=0;  higherq.qe=0; higherq.item[0]=ht;  /*根结点进高层队列*/
   lowerq.qa=-1;  lowerq.qe=0;  /*低层队列置空*/
```

```
    while (higherq.qa>=higherq.qe && lowerq.qa >= lowerq.qe)
    {  if(higherq.qa < higherq.qe) /*高层队列为空，则将低层队列赋给高层*/
       { higherq.qa=lowerq.qa;  higherq.qe=lowerq.qe;
         for(i=lowerq.qe;  i<=lowerq.qa; i++)
             higherq.item[i]=lowerq.item[i];
         lowerq.qa=-1;  lowerq.qe=0; /*将低层队列置空*/
    }
    while (higherq.qa>=higherq.qe)
    { p=higherq.item[higherq.qe ++];
      while(p!=NULL)
      {printf("%4l", p->data);
       if(p->fch!=NULL)  lowerq.item[++ lowerq.qa]=pfch; /*孩子结点进低层队列*/
       p=p->nsib;
      }
     }
   }
}
```

例题6-37 （中国人民大学2002年试题）

简述二叉哈夫曼树的建树方法。

【解答】（1）根据给定的n个权值$\{w_1, w_2, \cdots, w_n\}$构成$n$棵二叉树的集合$F=\{T_1, T_2, \cdots, T_n\}$，其中每棵二叉树$T_i$中只有一个带权为$w_i$的根结点，其左右子树均为空。

（2）在F中选取两棵根结点的权值最小的树作为左右子树构造一棵新的二叉树，且置新的二叉树的根结点的权值为其左、右子树上根结点的权值之和。

（3）在F中删除这两棵树，同时将新得到的二叉树加入F中。

（4）重复（2）和（3），直到F只含一棵树为止，这棵树就是哈夫曼树。

例题6-38 （中国人民大学2002年试题）

设计一算法，通过对一棵二叉树进行层次遍历求出一条从根到叶的最长路径，输出该路径上的结点序列。

（1）用二叉树的三叉链表为存储结构写出算法；

（2）简述若改用二叉链表为存储结构，算法应如何修改？

（3）简述一种不用层次遍历，求从根到叶的最长路径的其他算法。

【解答】

（1）

```
# define MAXN 50
struct node { int data;
              struct node *lc, *rc, *parent ;
            }
typedef struct node NODE;
      struct queue type { int qa, qe;  /*队列首、尾指针*/
                          NODE * item[MAXN]; /*队列数组*/
                        }
void longest path (NODE * bt)
{ struct queuetype queue;  /*定义一个队列；用于实现层次遍历*/
```

```
        NODE * lastenqueue;   /*记录最后进入队列的结点，从根结点到最后进入队列的结点的路
径
                              必须是一条最长路径*/
        NODE *path[MAXN];  /*记录最长路径*/
        NODE * p;
        int i;
        queue.qa=0; queue.qe=0;  queue.item[0]=bt;
        lastenqueue=bt;
        if(bt=NULL) {printf("Null tree!\n"); return;} /*空树则返回*/
        while (queue.qe<=queue.qa) /*层次遍历二叉树*/
        { p=queue.item[queue.qe++];
          if(p->ll!=Null)
          {queue.item[++queue.qa]=p->ll;
            lastenqueue=p->ll;
           }
           if (p->rl!=Null)
           { queue.item[++queue.qa]=p->rl;
              lastenqueue=p->rl;
           }
        }
        i=0; p=lastenqueue;
        do{ path[i++]=p;
           p=p->parent;
          }while(p!=bt)  /*记录最长的路径*/
        while(i>0)
            printf("%4d", path[--i] ->data;)
}
```

（2）若改为二叉链表，遍历部分不变，只需改变记录最长路径中求双亲结点的算法即可。

（3）可以定义一个数组用于记录每个结点的层次，采用深度优先遍历。比较数组元素找出一个最深的结点从根到该结点的路径即为最长路径。

例题6-39　（北京科技大学2002年试题）

设一棵三叉树中叶子结点数为n_0，度为2、3的结点数分别为n_2、n_3，试给出n_0与n_2、n_3之间的关系。

【解答】$n_2+2n_3=n_0-1$

例题6-40　（北京科技大学2002年试题）

算法填空题：

求Huffman树的带权路径长度（WPL）的算法如下，其中ht为树根结点的指针，S为工作栈，Clearstack(S)、Push(S，P)、Pop(S)和Emptystack(S)分别为置栈空、指针p进栈、出栈和判栈空的函数。请填写算法中下划线的空白之处，完成其功能。

```
Typedef float weight;
Typedef struct node
     { weight w;
```

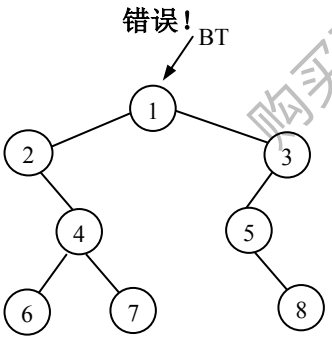
```
        struct node *Lchild, *Rchild, *parent;
    }hnode, *hlink
weight Hwpl(hlink ht)
    {hlink p;weight num=0.0;
        Clearstack(S);
          ①          ;
    while(! Emptystack(S))
    {  p=Pop(S);
          while(     ②     )
       { if(     ③     )num+=p->w;
          if(     ④     ) Push(S, p->Rchild);
          ⑤          ;
       }
    }
return(num);
}
```

【解答】

① Push(s, ht)
② p!=NULL
③ p->Rchild=NULL
④ p->Rchild!=NULL
⑤ p=p->Lchild

例题6-41 （北京科技大学2002年试题）

设一棵二叉树BT如下：



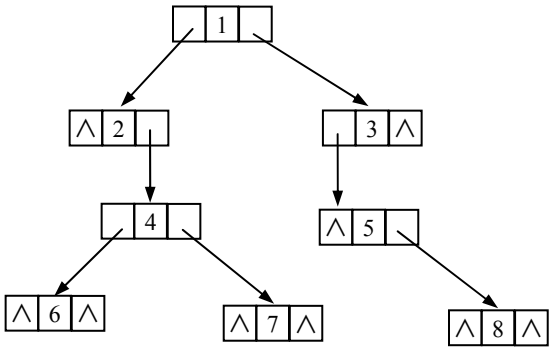
（1）请画出此二叉树BT的“顺序”及“二叉链表”式存储结构；

（2）写出按“先序”、“中序”和“后序”方法遍历二叉树BT所得到的结果序列，并画出BT的一棵后序线索二叉树。

【解答】

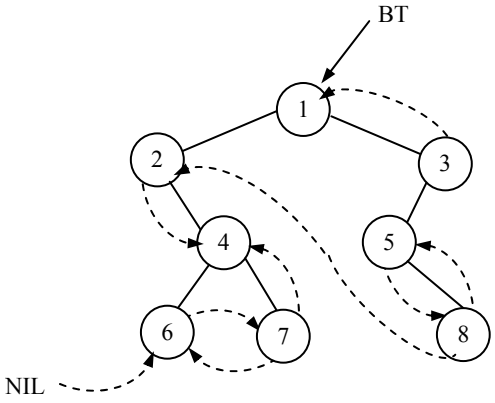
（1）顺序：1 2 3 0 4 5 0 0 0 6 7 0 8 0 0

二叉链表：



（2）先序：1　2　4　6　7　3　5　8
　　中序：2　6　4　7　1　5　8　3
　　后序：6　7　4　2　8　5　3　1

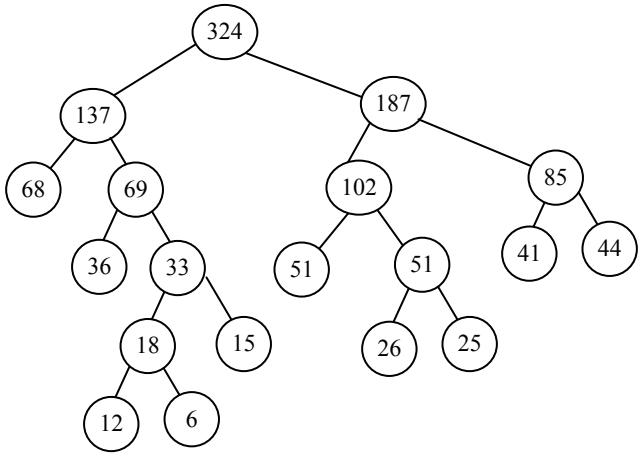
后序线索二叉树：



例题6-42 （北京科技大学2002年试题）

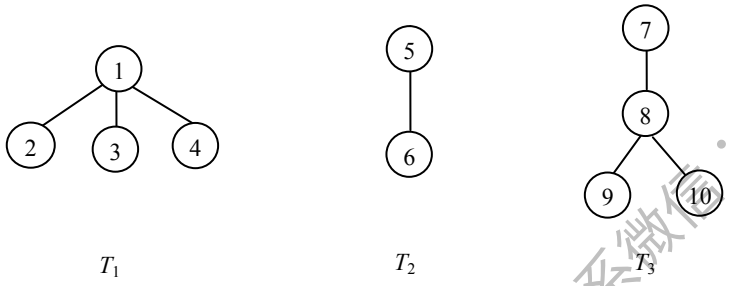
设记录的关键字（key）集合*K*={26，36，41，44，15，68，12，6，51，25}，以*K*为权值集合，构造一棵Huffman树，依次取*K*中各值，构造一棵二叉排序树。

【解答】Huffman树：



例题6-43 （北京科技大学2002年试题）

设森林$F=\{T_1, T_2, T_3\}$如下：



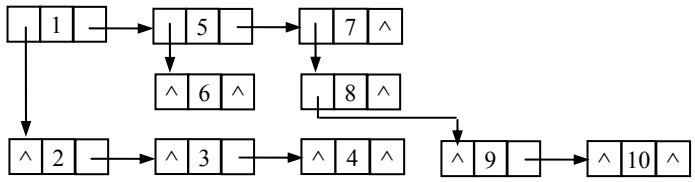
T_1　　T_2　　T_3

（1）若按“孩子兄弟表示法”存储此森林F，请画出其存储结构；

（2）写出按“先序”和“中序”方法遍历森林F所得到的结果序列。

【解答】

（1）



（2）先序：1　2　3　4　5　6　7　8　9　10

　　中序：2　3　4　1　6　5　9　10　8　7

例题6-44 （武汉理工大学2002年试题）

选择题：

已知一棵二叉树前序序列和中序序列分别为GFDBHCEA和DFHBGCAE，则该二叉树的后序序列为__A__，层次序列为__B__；若由森林转化得到的二叉树是非空的二叉树，则该二叉树__C__；如果满足条件__D__，则线索二叉树中结点p没有右儿子。

供选择的答案：

A、B：① DBHFEACG　② GFCDBEHA ③ DHBFAECG ④ DFGBCEHA

C：　① 根结点无右子树 ② 根结点可能有左子树和右子树

　　③ 根结点无左子树 ④ 各结点只有一个儿子

D：　① p↑.rchild=NULL ② p↑.rtag=1　③ p↑.rtag=0　④ p↑.rtag=NULL

【解答】A. ③　B. ②　C. ②　D. ③

例题6-45 （武汉理工大学2002年试题）

判断题：

高度为k的二叉树至多有2^{k-1}（$k\geq 1$）个结点。

【解答】错误。

例题6-46 （武汉理工大学2002年试题）

判断题：

Huffman树、平衡二叉树都是数据的逻辑结构。

【解答】正确。

例题6-47 （武汉理工大学2002年试题）

判断题：

Huffman树度为1的结点数等于度为2和0的结点数之差。

【解答】错误。

例题6-48 （武汉理工大学2002年试题）

算法设计题：

以二叉链为存储结构，求二叉树的深度。

【解答】设二叉树结点的结构为

```
struct BTreeNode {
    ElemType Data;
    struct BTreeNode *lchild;
    struct BTreeNode *rchild;
}
int BTrddDepth(BTreeNode *BT)
{  if (BT==NULL)return 0;
        else{
                int dep1=BTreeDepth(BTlchild);
                int dep2=BTreeDepth(BTrchild);
                if(dep1>dep2)return dep1+1;
                else return dep2+1;
           }
}
```

例题6-49 （南京理工大学2002年试题）

简答题：

写出表达式a*(b+c)-d/e*f的后缀表达式。

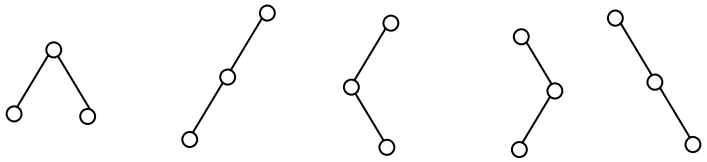
【解答】abc+*de/f*-

例题6-50 （南京理工大学2002年试题）

简答题：

画出有三个结点的所有二叉树。

【解答】



例题6-51 （南京理工大学2002年试题）

简答题：

写出一棵满k叉树上的叶子结点数和非叶子结点数之间的关系。

【解答】设叶子结点数为n_1，非叶子结点数为n_2，则$n=(k-1)n_2+1$

例题6-52 （南京理工大学2002年试题）

简答题：

有n个结点的二叉树，用二叉链表作为存储结构，空指针域有多少个？

【解答】$n+1$

例题6-53 （南京理工大学2002年试题）

选择题：

前序遍历和后序遍历结果相同的二叉树为（1）；

前序遍历和中序遍历结果相同的二叉树为（2）；

中序遍历和后序遍历结果相同的二叉树为（3）。

A. 一般二叉树

B. 根结点无左孩子的二叉树

C. 只有根结点的二叉树

D. 根结点无右孩子的二叉树

E. 所有结点只有左子树的二叉树

F. 所有结点只有右子树的二叉树

【解答】（1）C　　（2）F　　（3）E

例题6-54 （南京理工大学2002年试题）

选择题：

若以{4，5，6，3，8}作为叶子结点的权值构造哈夫曼树，则带权路径长度是_____。

A．55　　B．68　　C．59　　D．28

【解答】C

例题6-55 （南京理工大学2002年试题）

选择题：

在线索化二叉树中，结点*t*没有左子树的充要条件是_____。

A．t->lchild=null　　B．t->ltag=1

C．t->ltag=1且t->lchild=null　　D．以上都不对

【解答】B

例题6-56 （南京理工大学2002年试题）

选择题：

树有先根遍历和后根遍历，树可以转化为对应的二叉树。下面的说法正确的是___。

A．树的后根遍历与其对应的二叉树的后根遍历相同

B．树的后根遍历与其对应的二叉树的中根遍历相同

C．树的先根遍历与其对应的二叉树的中根遍历相同

D．树的先根遍历与其对应的二叉树的先根遍历相同

【解答】D

例题6-57 （南京理工大学2002年试题）

二叉树中序遍历的非递归算法如下所示。请填写算法中下划线的空白处。

```
Status Inorder(BiTree T){
    InitStack(s);push(s, T);
    While (   (1)   ){
          While (gettop(s, p)&&p) push (s,   (2)   );
          pop(s, p);
          if(!StackEmpty(s)){
               pop(s, p);printf(   (3)   );
               push(s,   (4)   );
          }//if
    }//while
    return ok;
}//Inorder
```

注：

InitStack(s);初始化一个栈s；

push(s，p);将所指向的结点进s栈；

pop(s，p);s栈顶元素出栈；
gettop(s，p);取s栈顶元素；
StackEmpty(s);判栈s是否为空。

【解答】
（1）!StackEmpty(s)　（2）p->lchild　（3）"%d"，p->data　（4）p->rchild

6.5　自 测 题

自测题6-1

设高为h的二叉树只有度为0和2的结点，则此类二叉树的结点数至少为____，至多为_____。

A. $2h$　　B. $2h-1$　　C. $2h+1$　　D. $h+1$
E. 2^{h-1}　　F. $2^{h}-1$　　G. $2^{h+1}+1$　　H. $2^{h}+1$

自测题6-2

一棵有124个叶子结点的完全二叉树，最多有_____个结点。

A. 247　　B. 248　　C. 250　　D. 251

自测题6-3

以二叉链表作存储结构，编写计算二叉树中叶子结点数目的递归函数。

自测题6-4

任给定一二叉树表示的算术表达式（存储结构为二叉链表），试编写一个算法，由该二叉树输出该表达式，若原表达式有括号亦应加上。

自测题6-5

一个深度为h的满m叉树有如下性质：第h层上的结点都是叶子结点，其他各层上每个结点有m棵非空子树。问：

（1）第k层最多有多少个结点？（$k\leqslant h$）

（2）整棵树最多有多少个结点？

（3）若按层次从上到下，每层从左到右的顺序从1开始对全部结点编号，编号为i的结点的双亲结点的编号是什么？编号为i的结点的第j个孩子结点（若存在）的编号是什么？

自测题6-6

已知中序线索二叉树采用二叉链表存储结构，链结点的构造为：

lbit	lchild	data	rchild	rbit

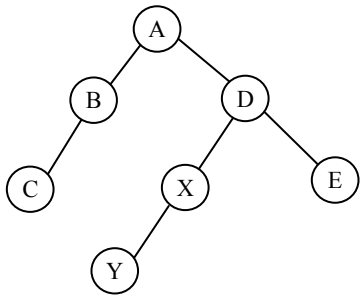
其中lbit位为0，则lchild为指向结点的前驱，否则lchild为指向左孩子结点；rbit位为0，则rchild为指向结点的后继，否则rchild为指向右孩子结点。下面的算法返回x所指结点的直接后继结点的位置。若该算法有错，则请改正错误；若无错，不作任何修改，并请写“正确”二字。

```
FUNCTION INSUCC(x)
BEGIN
    s:=rchild(x)
    IF rbit(s)<>0 THEN
        WHILE lbit(s)<>0 DO
        BEGIN
            s:=rchild(s)
        END
    return(s)
END
```

自测题6-7

将下图的二叉树按中序线索化，结点X的右指针和Y的左指针分别指向_____。

（1）A，D　（2）B，C　（3）D，A　（4）C，A



自测题6-8

已知深度为h的二叉树以一维数组BT(1:2^h-1)作为其存储结构。请写一算法，求该二叉树中叶子结点的个数。

自测题6-9

如果两棵二叉树的结构相同，并且结点中的数值都相等，则这两棵二叉树相等。下面的函数过程equal描述了判断二叉树是否相等的算法。请将{? i}处分别替换为必要的成份。

```
TYPE
        nodeptr=^node;
        node=RECORD
        info:CHAR;
        left, right:nodeptr;
    END;
    element=RECORD
        t1:nodeptr;
        t2:nodeptr
```

```
    END;
    stack=RECORD
        item:ARRAY[1..max]OF element;
        top:0..max
    END;
VAR s:stack;

PROCEDURE in it(VAR s:Stack);
FUNCTION empty(s:stack):boolean;
PROCEDURE push(VAR s:stack;  x:element);
PROCEDURE pop(VAR s:stack;VAR t:element);
FUNCTION equal(p, q:nodeptr):boolean;
    VAR
        ok, sempty:boolean;
        x:element;
BEGIN
    init(s);
    ok:=true;
    REPEAT
        WHILE(p<>NIL)AND(q<>NIL)AND ok DO
            IF p^.info =q^.info THEN
                BEGIN
                    x.t1:=p;
                    x.t2:=q;
                    push(s, x);
                    p:=p^.left;
                    q:=q^.left;
                END
            ELSE ok:=false;
        IF{?(1)}THEN ok:=false;
        sempty:=empty(s);
        IF ok AND NOT empty THEN
        BEGIN
            pop(s, x);
            p:=x.t1;
            q:=x.t2;
            { ?(2)  };
        END
    UNTIL {  ?(3)  };
    IF NOT empty(s) THEN init(s);
    equal:=ok
END;
```


自测题6-10

设二叉树的结点结构是：

LC	data	RC

其中LC和RC分别为指向左、右子树的指针，data是字符型数据。

试写出算法，求任意二叉树中第一条最长的路径长度，并输出此路径上各结点的值。

自测题6-11

回答问题并写出推导过程：

含N个结点和N个叶子结点的完全三叉树的高度是多少？

自测题6-12

有二叉树中序序列为：ABCEFGHD

后序序列为：　　ABFHGEDC

请画出此二叉树。

自测题6-13

填空：

具有n个结点的满二叉树，其叶子结点的个数为_____。

自测题6-14

判断正误：

完全二叉树的某结点若无左孩子，则它必是叶子结点。

自测题6-15

由二叉树的前序和后序遍历序列_____惟一地确定这棵二叉树。

A. 能　　B. 不能

自测题6-16

在下列3种次序的线索二叉树中，_____对查找指定结点在该次序下的后继效果较差。

A. 前序线索树　　B. 中序线索树　　C. 后序线索树

自测题6-17

已知一个二叉树的前序及中序遍历结果，请写一算法，恢复该二叉树。

自测题6-18

已知某电文中共出现了10种不同的字母，每个字母出现的频率分别为A：8，B：5，C：3，D：7，E：7，F：23，G：9，H：11，I：2，J：35，现在对这段电文用三进制进行编码（即码字由0，1，2组成），问电文编码总长度至少有多少位？请画出相应的图。

自测题6-19

已知某字符串S中共有8种字符，各种字符分别出现2次、1次、4次、5次、7次、3次、4次和9次，对该字符串用{0，1}进行前缀编码，问该字符串的编码至少有多少位？

自测题6-20

具有n个叶子的二叉树，每个叶子的权值为$W_i(1\leqslant i\leqslant n)$其中带权路径长度最小的二叉

树被称为_____。

自测题6-21

如果只考虑有序树的情形，那么具有7个结点的不同形态的树共有_____。

A. 132　　B. 154　　C. 429　　D. 前三者均不正确

自测题6-22

具有7个结点的互不相似的二叉树共有多少棵？

自测题6-23

缩写递归算法，依据树的表示法及其根结点创建树的孩子-兄弟链表存储结构，要求写算法以前先写出这两种存储结构的类型说明。

6.6　自测题答案

【自测题6-1】

因为此二叉树中只有度为0和2的结点，则结点数最少的情况是除根结点外，每一层都只有两个结点，故最少共有$2(h-1)+1=2h-1$个结点；结点数最多的情况是此二叉树是完全二叉树，且第h层有$2h-1$个结点，故最多共有$1+2+4+\cdots+2^{h-1}=2^h-1$个结点。

答案依次为B，F。

【自测题6-2】

答案为B。

【自测题6-3】

```
TYPE
        bitree=^binode
        binode=RECORD
            key:keytype;
            lchild, rchild:bitree;
        END;

FUNCTION calleaf(VAR t:bitree):integer;
BEGIN
        IF t=NIL THEN
            return(0);
        ELSE IF(t^.lchild=NIL)AND(t^.rchild=NIL)
            THEN return(1);
            ELSE
                return(leaf(t^.lchild)+leaf(t^.rchild));
            ENDIF
        ENDIF
END
```

【自测题6-4】

由表达式树输出表达式实际上是一个中序遍历的过程，同时要判断运算符的优先级，若左右子树运算符的优先级小于父结点，则要加括号。

答案：

```
typedef struct node{
        struct node   *left, *right;
        OPTYPE    op;      //操作符
}NODE, *TREE;

int compare(OPTYPE op1, OPTYPE op2);
//比较两个操作符的优先级
//op1高于op2返回1
//op1等于op2返回0
//op1低于op2返回-1
{
…  //具体实现依赖于实际操作符的优先级
}

void printext(TREEt)
{
        int brack;
        if(t->left){
            brack=(compare(t->op, t->left->op)==1);
            if(brack)printf("(");
            printexp(t->left);
            if(brack)print(")");
        }
        print(t->op);
        if(t->right){
            brack=(compare(t->op, t->right->op)!=-1);
            if(brack)printf("(");
            printexp(t->right);
            if(brack)print(")");
            }
 }
```

【自测题6-5】

（1）第k层上最多有m^{k-1}个结点。

（2）整棵树最多有$1+m+m^2+\cdots+m^{h-1}=\dfrac{1}{m-1}(m^n-1)$个结点。

（3）编号为i的结点的双亲结点为$\left\lfloor\dfrac{i+m-2}{m}\right\rfloor$。

编号为i的第j个子结点的编号为$m(i-1)+j+1$。

【自测题6-6】

错误，修改如下：

```
FUNCTION INSUCC(x)
```

```
BEGIN
    s:=rchild(x)
    IF rbit(x)<>0 THEN
        WHILE lbit(s)<>0 DO
        BEGIN
            s:=lchild(s)
        END
    return(s)
END
```

【自测题6-7】

X的右指针指向X的中序后继。Y的右指针指向Y的中序后继。

答案选（3）。

【自测题6-8】

数组BT的值为在满二叉树中编号为数组下标值的结点值。若该结点不存在，则数组值取0。

```
FUNCTION LEAFCount(BT, h)
VAR
    i, len, count:   INTEGER;
BEGIN
    len:=2^h-1;
    count:=0;
    FOR i:=1 TO len DO
        IF BT[i]<>0 THEN
            IF i*2>len THEN
                count:=count+1
            ELSE IF BT[i*2+1]=0 AND BT[i*2]=0 THEN
                count:=count+1;
    return(count);
END
```


【自测题6-9】

判断是否相等的过程实际上是一个先序遍历的过程，可以用递归来实现。这里采用栈的方式模拟递归实现。

（1）p<>NIL OR q<>NIL

（2）p:=p^.right;q:=q^.right

（3）NOT ok OR sempty

【自测题6-10】

先求各个结点的深度。

通过某一结点的最长路径为该结点的左子树深度+右子树深度+1。

```
typedef struct node{
    struct node   *LC, *RC;
    DataType       data;
}NODE, *PNODE;     //二叉树结点
typedef struct enode{
```

```
    struct enode   *LC, *RC;
    DataType        data;
    int               len;        //深度
    int               flag;       //flag=0左子树深度更大
                                  //flag=1右子树深度更大
    }enode, *PENODE;              //扩展的二叉树结点

int maxlen=0;      //最长的路径长度
PENODE    maxpen=NULL;    //最长路径的结点

PENODE create(PNODE pn){
    //根据初始二叉树，创建扩展二叉树，并计算各结点的深度
    PNODE  pen;
    int   LL, RL;       //左右子树的深度

    if(pn==NULL)return NULL;
    pen=malloc(sizeof(enode));
    pen->LC=create(pn->LC);
    pen->RC=create(pn->RC);

    //计算深度
    LL=(pen->LC==NULL)?0:pen->LC->len;
    RL=(pen->RC==NULL)?0:pen->RC->len;
    if(LL>RL)  {
        pen->flag=0;
        pen->len=LL+1;}
    else{
        pen->flag=1;
        pen->len=RL+1;
    }

    //求最长路径
    if(LL+RL+1>maxlen){
        maxlen=LL+RL+1;
        maxpen=pen;
    }
    return pen;
}
void revdisplay(PENODE p){
    if(!p)return;
    if(!p->flag)
        revdisplay(p->LC);
    else
        revdisplay(p->RC);
    print(p->data);
}

void destroy(PENODE P){
    if(!p)    return;
    destroy(p->LC);
```

```
    destroy(p->RC);
    free(p);
}

void main(){
    PENODE p;
    PNODE p0;
    …    //初始二叉树的创建
    p=create(p0);
    print(maxlen);    //显示最长路径的长度
    revdisplay(maxpen->LC)  //显示最长路径左边部分
    while(!maxpen){              //显示最长路径右边部分
        print(maxpen->data);
        if(!p->flag)  maxpen=maxpen->LC;
        else maxpen=maxpen->RC;
    }
    destroy(p);
}
```

【自测题6-11】

（1）含N个结点的完全三叉树的高度设为H

$1+3+\cdots+3^{H-2}+1\leqslant N\leqslant 1+3+\cdots+3^{H-1}$

$(3^{H-1}-1)/2+1\leqslant N\leqslant (3^{H}-1)/2$

$3^{H-1}\leqslant 2N-1\leqslant 3^{H}-2$

所以$H=\lfloor \log_3 2n-1 \rfloor+1$

（2）含N个叶子结点的完全三叉树的高度设为H

$3^{H-2}\leqslant N\leqslant 3^{H-1}$

所以

$$H=\begin{cases}\log_3 N+1 \text{或} \log_3 N+2 & N\text{为3的乘幂}\\ \lfloor \log_3 N \rfloor+2 & N\text{不为3的乘幂}\end{cases}$$

【自测题6-12】

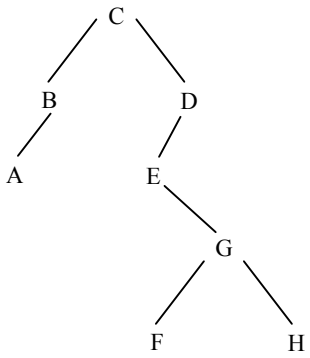
此类题目存在固定解法。首先根据后序序列找到根结点，然后分别得到左右子树的中序序列和后序序列，如此递归。

答案：根据后序序列知根结点为C。

因此左子树：中序序列为AB，后序序列为AB。

右子树：中序序列为EFGHD，后序序列为FHGED。

依次推得该二叉树如下图所示。



【自测题6-13】

设满二叉树的深度为h，则

总的结点个数n=1+2+4+…+2^{h-1}

叶子结点个数为2^{h-1}=(n+1)/2

答案：(n+1)/2

【自测题6-14】

完全二叉树的定义为：

深度为k，有n个结点的二叉树，当且仅当其每一个结点都与深度为k的满二叉树中编号从1到n的结点一一对应时，称为完全二叉树。

根据此定义，若完全二叉树中某结点无左孩子，则其必定没有右孩子，因此是叶子结点。

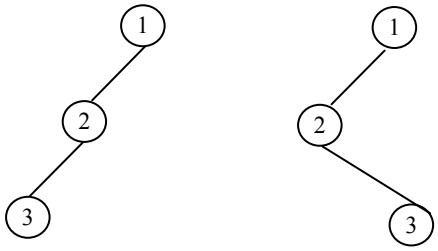
答案：正确。

【扩展】

注意满二叉树的定义为：一棵深度为k且有2^k-1个结点的二叉树称为满二叉树。

【自测题6-15】

如下图所示的两棵二叉树，其前序遍历序列都为123，后序遍历序列都为321，但两者显然不相同。



前，后序遍历相同的两棵二叉树

答案为B。

【扩展】

由二叉树的前序和中序遍历序列即可惟一地确定这棵二叉树。
由二叉树的后序和中序遍历序列即可惟一地确定这棵二叉树。

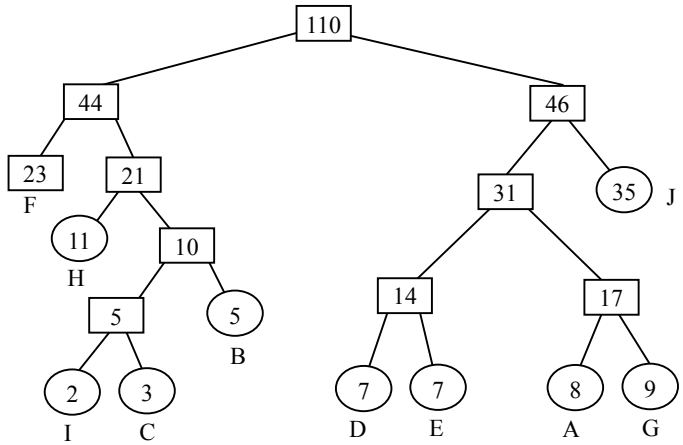
【自测题6-16】
答案为C。

【自测题6-17】

```
FUNCTION restore(in) /*in is a string*/
BEGIN
     len:=strlen(in);
     IF(len=0)return(NIL);
     i:=i+1;
     IF(len=1)THEN
     BEGIN
         new(p);
         p^.data:=substr(in, 1, 1);
         p^.left:=p^.right:=NIL;
     END
     ELSE BEGIN
         r:=substr(preorder, i, 1);
         new(p);
         p^.data:=r;
         j:=index(in, r);
         p^.left:=restore(substr(in, 1, j-1));
        p^.right:=restore(substr(in, j+1, len-j));
     END
     return(p);
END

BEGIN  /*main*/
     i:=0;
     root:=restore(inorder);
END;
```

在上面的程序中，inorder和proder分别是二叉树的中序和前序遍历序列的字符串，函数restore中的i是main中定义的i。

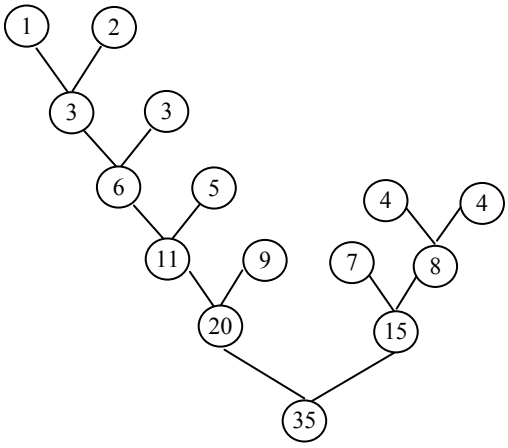
【自测题6-18】
相应的哈夫曼编码树如下图所示。



码长至少为：(23+35)×2+11×3+(5+14+17)×4+5×5=318。

【自测题6-19】

对8种字符根据其相对频率构造的哈夫曼树如下图所示。



该字符串的编码至少有5×1+5×2+4×3+3×4+3×4+3×5+9×2+7×2=98位。

【自测题6-20】

哈夫曼树/最优二叉树。

【自测题6-21】

具有n个结点有不同形态的树的数目t_n和具有n-1个结点互不相似的二叉树的数目相同。具有n个结点互不相似的数目为$\frac{1}{n+1}C_{2n}^{n}$。

$$t_6=\frac{1}{6+1}C_{12}^{6}=132$$

答案为A。

【自测题6-22】

具有n个结点互不相似的二叉树的数目为$\frac{1}{n+1}C_{2n}^{n}$。

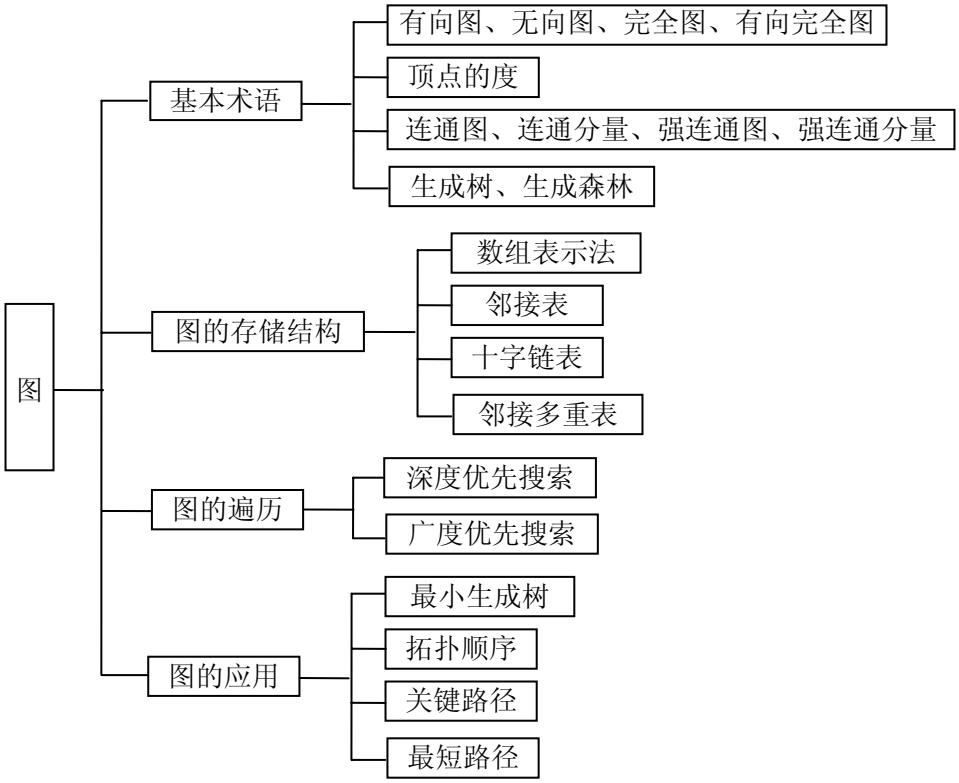
故具有7个结点互不相似的二叉树的数目为$\frac{1}{7+1}C_{2\times 7}^{7}=429$个。

【自测题6-23】

```
CONST maxsize=…{结点的最大个数}
TYPE PTNODE=RECORD
         data    :DataType;
         parent  :INTEGER;
     END
     PTREE=RECORD
         Nodes:ARRAY[1.maxsize] of PTNODE;{结点}
         n:integer;{结点个数}
     END;
     CSTREE=^CSNODE;
     CSNODE=RECORD
         data  :DataType;
         firstchild, nextsibling:CSTREF;
     END
FUNCTION PT2CS(pt:PTREE;root:integer):CSTREE
{根据双亲表示法表示的树转换成孩子—兄弟链表结构表示的树}
VAR
     ct:  CSTREE;
     child, sibling:CSTREE;
     i:integer;
     isFirst:  boolean;
BEGIN
     new(ct);
     ct^.data:=pt.nodes[root].data;
     ct^.firstchild:=null;
     ct^.nextsibling:=null;
     isFirst:=TRUE;
     FOR i:=1 TO pt.n DO
         IF  pt.nodes[i].parent=root THEN
         BEGIN
             IF  isFirst THEN BEGIN
             BEGIN
                 IF  isFirst THEN BEGIN
                         child:=PT2CS(pt, i);
                         ct^.firstchild:=child;
                         isFirst:=FALSE;
                         END
             ELSE   BEGIN
                 sibling:=PT2CS(pt, i);
                 child^.nextsibling:=sibling;
                 child:=sibling;
                 END
         END
         PT2CS:=ct;
END;
```

第 7 章　图

7.1　基本知识结构图



7.2　知 识 点

1. 基本概念

图有有向图和无向图两种，顶点间的关系是有方向的图称为有向图，否则称为无向图。有$\frac{1}{2}n(n-1)$条边的无向图称为完全图，具有$n(n-1)$条弧的有向图称为有向完全图。

如果对于图中任意两个顶点都是连通的，则称该图是连通图。无向图中的极大连通子

图称为该图的连通分量。在有向图中，如果对于每一对v_i，$v_j \in V$，$v_i \neq v_j$，从v_i到v_j和从v_j到v_i都存在路径，则称该图是强连通图。有向图中的极大强连通子图称作有向图的强连通分量。

一个连通图的生成树是一个极小连通子图，它含有图中全部顶点，但只有足以构成一棵树的$n-1$条边。一个有向图的生成森林由若干棵有向树组成，含有图中全部顶点，但只有足以构成若干棵不相交的有向树的弧。

2. 图的存储结构

图的存储结构包括数组表示法、邻接表、十字链表和邻接多重表。

数组表示是用两个数组分别存储数据元素（顶点）的信息和数据元素之间的关系（边或弧）的信息。

在邻接表中对图中每个顶点建立一个单链表，第i个单链表中的结点表示依附于顶点v_i的边（对有向图是以顶点v_i为尾的弧）。每个结点由3个域组成，其中邻接点域指示与顶点v_i邻接的点在图中的位置，链域指示下一条边或弧的结点，数据域存储和边或弧相关的信息。

十字链表也是一种链式存储结构，可以看成是将有向图的邻接表和逆邻接表结合起来得到的一种链表。在十字链表中，对应于有向图中每一条弧有一个结点，对应于每个顶点也有一个结点。

邻接多重表是无向图的一种链式存储结构，它的结构与十字链表类似，每一条边用一个结点表示，该结点由五个域组成，其中标志域用以标记该条边是否被搜索过，有两个域用于记录该边依附的两个顶点在图中的位置；另两个域分别指向依附于该边的两个顶点的下一条边。每个顶点也用一个结点表示，每个结点由两个域组成，分别存储和该顶点相关的信息和第一条依附于该顶点的边。

3. 图的遍历

从图中某一顶点出发访遍途中其余顶点，且使每一个顶点仅被访问一次，这一过程叫图的遍历。有深度优先搜索和广度优先搜索两种。

4. 最小生成树

对于n个顶点的连通网可以建立许多不同的生成树，每一棵生成树都可以是一个通信网，选择使总的通信耗费最少的生成树，这个问题就是构造连通网的最小代价生成树（简称最小生成树）的问题，一棵生成树的代价就是树上各边的代价之和。

构造最小生成树的算法大多都利用了最小生成树的MST性质，主要有普里姆算法和克鲁斯卡尔算法。

5. 有向无环图及其应用

一个无环的有向图称作有向无环图。有向无环图可以用来描述一项工程或系统的进行过程的有效工具。对整个工程和系统，人们关心的是两个方面的问题：一是工程能否顺利进行；二是估算整个工程完成所必需的最短时间。对应于有向图，即为进行拓扑排序和求

关键路径的操作。

6. 最短路径

有两种最常见的最短路径问题，一是从某源点到其余各顶点的最短路径问题，二是每一对顶点之间的最短路径问题。单源点的最短路径问题可以用迪杰斯特拉（Dijkstra）提出的按路径长度递增的次序产生最短路径的算法。求每一对顶点之间的最短路径可以每次以一个顶点为源点，重复执行迪杰斯特拉算法，也可以采用弗洛伊德（Floyd）算法。

7.3　习题及参考答案

1．已知下面的有向图，请给出该图的

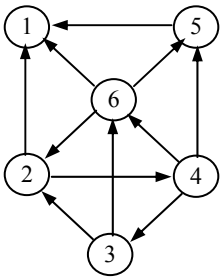
（1）每个顶点的入/出度；

（2）邻接矩阵；

（3）邻接表；

（4）逆邻接表；

（5）强连通分量。



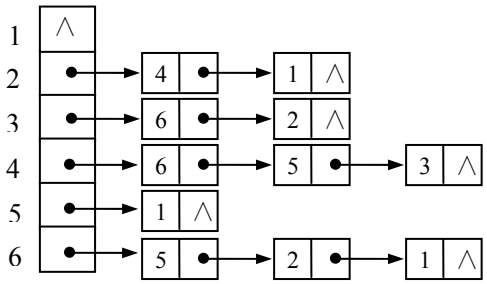
【解答】（1）每个顶点的入/出度

顶点	1	2	3	4	5	6
入度	3	2	1	1	2	2
出度	0	2	2	3	1	3

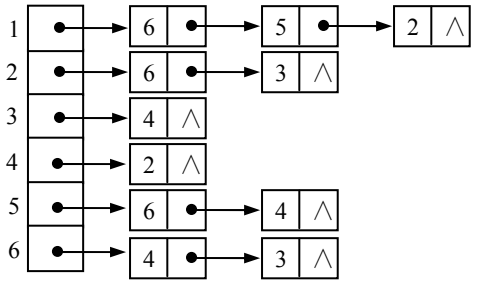
（2）邻接矩阵

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 \end{pmatrix}$$

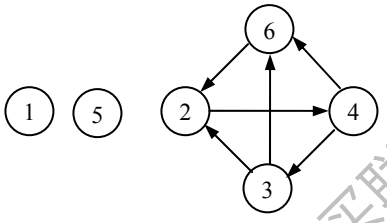
（3）邻接表



（4）逆邻接表



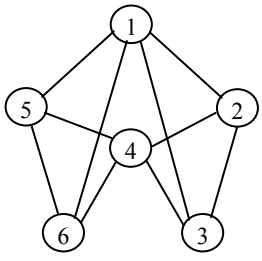
（5）有3个强连通分量，如下图所示。



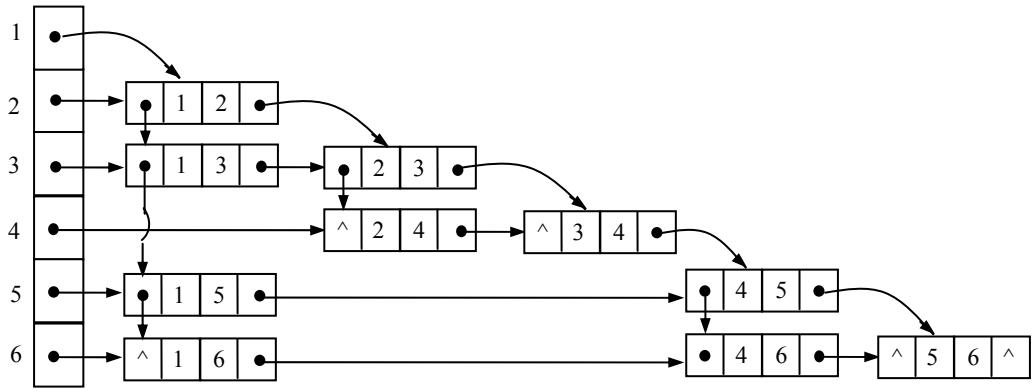
2．已知有向图的邻接矩阵为$A_{n\times n}$，试问每一个$A_{n\times n}^{(k)}$（k=1，2，…，n）各具有何种实际含义？

【解答】记（$a_{ij}^{(k)}$）=$A_{n\times n}^{k}$，则$a_{ij}^{(k)}$为由i到j的长度为k的路径数。注意，不能理解为简单路径。

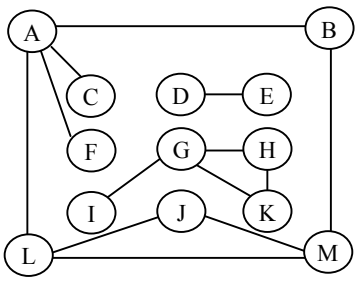
3．画出下面的无向图的邻接多重表，使得其中每个无向边结点中第一个顶点号小于第二个顶点号，且每个顶点的各邻接边的链接顺序，为它所邻接到的顶点序号由小到大的顺序。列出深度优先和广度优先搜索遍历该图所得顶点序列和边的序列。



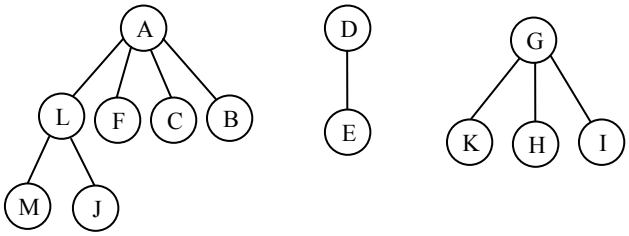
【解答】



4．试对下图所示的无向图画出其广度优先生成森林。



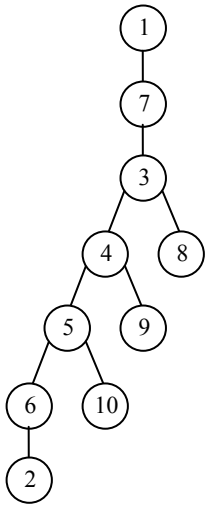
【解答】从顶点A出发进行广度优先遍历所得广度优先生成森林为：

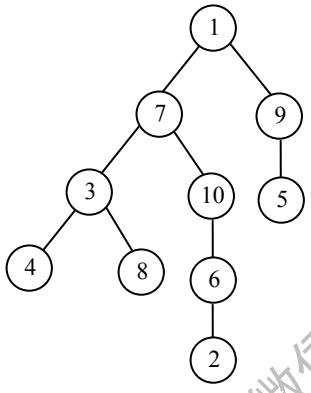


5．已知以二维数组表示的图的邻接矩阵如下图所示。试分别画出自顶点1出发进行遍历所得的深度优先生成树和广度优先生成树。

	1	2	3	4	5	6	7	8	9	10
1	0	0	0	0	0	0	1	0	1	0
2	0	0	1	0	0	0	1	0	0	0
3	0	0	0	1	0	0	0	1	0	0
4	0	0	0	0	1	0	0	0	1	0
5	0	0	0	0	0	1	0	0	0	1
6	1	1	0	0	0	0	0	0	0	0
7	0	0	1	0	0	0	0	0	0	1
8	1	0	0	1	0	0	0	0	1	0
9	0	0	0	0	1	0	1	0	0	1
10	1	0	0	0	0	1	0	0	0	0

【解答】深度优先生成树　　广度优先生成树

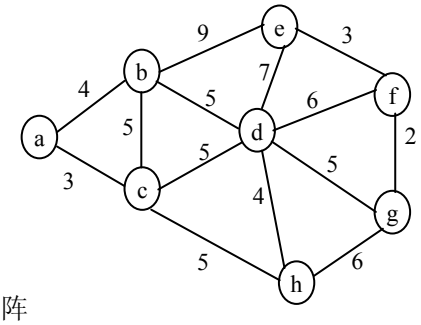




6. 请对下面的无向带权图，

（1）写出它的邻接矩阵，并按普里姆算法求其最小生成树；

（2）写出它的邻接表，并按克鲁斯卡尔算法求其最小生成树。

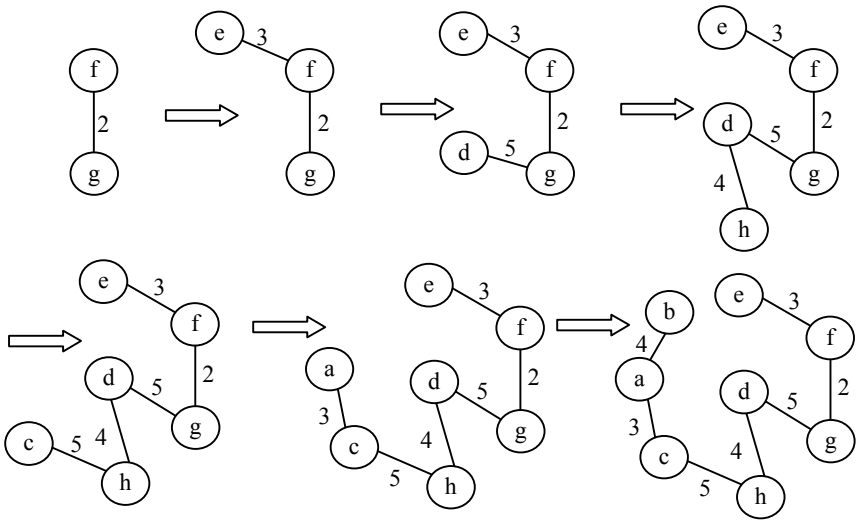


【解答】（1）邻接矩阵

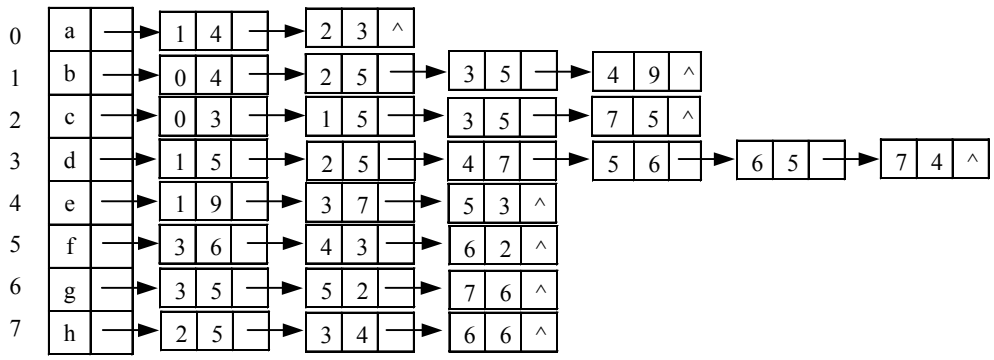
a　b　c　d　e　f　g　h

$$
\begin{array}{c}
\\
a\\
b\\
c\\
d\\
e\\
f\\
g\\
h
\end{array}
\begin{bmatrix}
0 & 1 & 1 & 0 & 0 & 0 & 0 & 0\\
1 & 0 & 1 & 1 & 1 & 0 & 0 & 0\\
1 & 1 & 0 & 1 & 0 & 0 & 0 & 1\\
0 & 1 & 1 & 0 & 1 & 1 & 1 & 1\\
0 & 1 & 0 & 1 & 0 & 1 & 0 & 0\\
0 & 0 & 0 & 1 & 1 & 0 & 1 & 0\\
0 & 0 & 0 & 1 & 0 & 1 & 0 & 1\\
0 & 0 & 1 & 1 & 0 & 0 & 1 & 0
\end{bmatrix}
$$

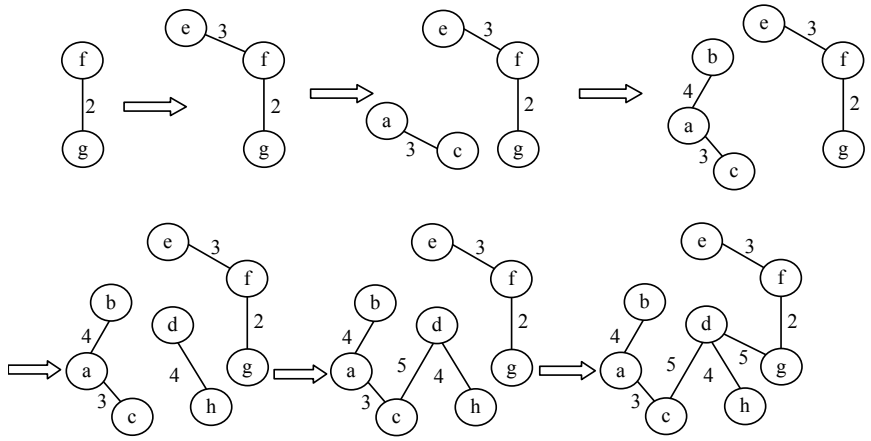
按普里姆算法求得最小生成树为：



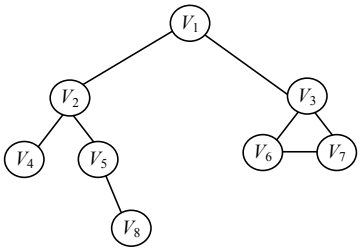
（2）邻接表



按克鲁斯卡尔算法求得最小生成树为：



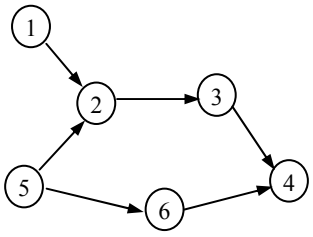
7.试对下图所示无向图执行求关结点的算法,分别求出每个顶点的visited[i]和low[i]值，i=1，2，…，vexnum。



【解答】

G.vertices[i].data	V_1	V_2	V_3	V_4	V_5	V_6	V_7	V_8
visited[i]	1	2	6	3	5	7	8	4
low[i]的产生序号	1	4	7	3	1	6	5	2
low[i]	1	1	1	2	2	6	6	2

8.试列出下图中全部可能的拓扑有序序列，并指出应用拓扑排序求得的是哪一个序列（注意：应先确定其存储结构）。



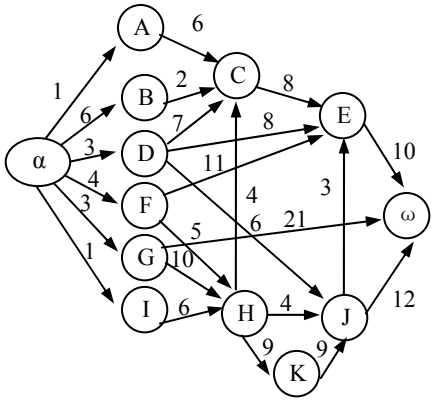
【解答】注意养成系统化思维方法：

5	6	1			2			3			4

5		1	6		2			3			4
5		1			2	6		3			4
5		1			2			3	6		4
		1	5	6	2			3			4
		1	5		2	6		3			4
		1	5		2			3	6		4

其中，第一个序列为应用拓扑排序所求得的序列。

9．对于图中所示的AOE网络，计算各活动弧的e(a_i)和l(a_j)函数值、各事件（顶点）的ve(v_i)和vl(v_j)函数值；列出各条关键路径。



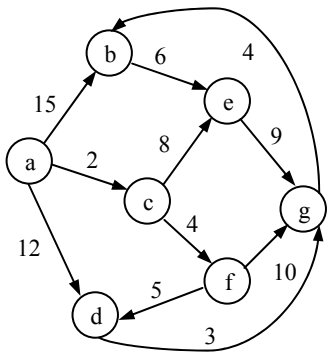
【解答】

顶点	ve	vl
α	0	0
A	1	20
B	6	24
C	17	26
D	3	19
E	34	34
F	4	8
G	3	3
H	13	13
I	1	7
J	31	31
K	22	22
ω	44	44

边	e	j	j-e
(α, A)	0	19	19
(α, B)	0	18	18
(α, D)	0	16	16
(α, F)	0	4	4
(α, G)	0	0	0
(α, I)	0	6	6
(A, C)	1	20	19
(B, C)	6	24	18
(D, C)	3	19	16
(D, E)	3	26	23
(D, J)	3	25	22
(F, E)	4	23	19
(F, H)	4	8	4
(G, ω)	3	23	20
(G, H)	3	3	0
(I, H)	1	7	6
(C, E)	17	26	9
(H, C)	13	22	9
(H, J)	13	27	14
(H, K)	13	13	0
(K, J)	22	22	0
(J, E)	31	31	0
(J, ω)	31	32	1
(E, ω)	34	34	0

关键路径只有一条：（α，G，H，K，J，E，ω）

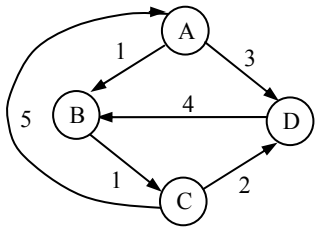
10. 试利用Dijkstra算法求下图中从顶点a到其他各顶点间的最短路径，写出执行算法过程中各步的状态。



【解答】从顶点a到其他各点的最短路径的求解过程如下：

终点 / Dist	b	c	d	e	f	g	S（终点集）
K=1	15 (a，b)	**2 (a，c)**	12 (a，d)				{a，c}
K=2	15 (a，b)		12 (a，d)	10 (a，c，e)	**6 (a，c，f)**		{a，c，f}
K=3	15 (a，b)		11 (a，c，f，d)	**10 (a，c，e)**		16 (a，c，f，g)	{a，c，f，e}
K=4	15 (a，b)		**11 (a，c，f，d)**			16 (a，c，f，g)	{a，c，f，e，d}
K=5	15 (a，b)					**14 (a，c，f，d，g)**	{a，c，f，e，d，g}
K=6	**15 (a，b)**						{a，c，f，e，d，g，b}

11．试利用Floyd算法求图中所示的有向图中各对顶点之间的最短路径。



【解答】

A	$A^{(0)}$				$A^{(1)}$			
	1	2	3	4	1	2	3	4
1	0	1	∞	3	0	1	∞	3
2	∞	0	1	∞	∞	0	1	∞
3	5	∞	0	2	5	**6**	0	2

4	∞	4	∞	0	∞	**4**	∞	0
PATH	PATH $^{(0)}$				PATH $^{(1)}$			
	1	2	3	4	1	2	3	4
1		AB		AD		AB		AD
2			BC				BC	
3	CA			CD	CA	**CAB**		CD
4		DB				DB		
A	A $^{(2)}$				A $^{(3)}$			
	1	2	3	4	1	2	3	4
1	0	1	**2**	3	0	1	2	3
2	∞	0	1	∞	**6**	0	1	**3**
3	5	6	0	2	5	6	0	2
4	∞	4	**5**	0	**10**	4	5	0
PATH	PATH $^{(2)}$				PATH $^{(3)}$			
	1	2	3	4	1	2	3	4
1		AB	**ABC**	AD		AB	ABC	AD
2			BC		**BCA**		BC	**BCD**
3	CA	CAB		CD	CA	CAB		CD
4		DB	**DBC**		**DBCA**	DB	DBC	

注：A $^{(4)}$和A $^{(3)}$相同；PATH $^{(4)}$和PATH $^{(3)}$相同。

12．编写算法，由依次输入的顶点数目、弧的数目、各顶点的信息和各条弧的信息建立有向图的邻接表。

【解答】算法如下：

```
Status Build_AdjList(ALGraph &G)//输入有向图的顶点数，边数，顶点信息和边的信息建立邻接表
{
  InitALGraph(G);
  scanf("%d", &v);
  if(v<0) return ERROR; //顶点数不能为负
  G.vexnum=v;
  scanf("%d", &a);
  if(a<0) return ERROR; //边数不能为负
  G.arcnum=a;
  for(m=0;m<v;m++)
    G.vertices[m].data=getchar(); //输入各顶点的符号
  for(m=1;m<=a;m++)
  {
    t=getchar();h=getchar(); //t为弧尾，h为弧头
    if((i=LocateVex(G, t))<0) return ERROR;
    if((j=LocateVex(G, h))<0) return ERROR; //顶点未找到
```

```
    p=(ArcNode*)malloc(sizeof(ArcNode));
    if(!G.vertices.[i].firstarc) G.vertices[i].firstarc=p;
    else
    {
      for(q=G.vertices[i].firstarc;q->nextarc;q=q->nextarc);
      q->nextarc=p;
    }
    p->adjvex=j;p->nextarc=NULL;
  }//for
  return OK;
}//Build_AdjList
```

13. 试在邻接矩阵存储结构上实现图的基本操作：InsertVex(G，v)，InsertArc(G，v，w)，DeleteVex(G，v)和DeleteArc(G，v，w)。

【分析】本题中的图G均为有向无权图，其余情况容易由此写出。

【解答】算法如下：

```
Status Insert_Vex(MGraph &G,  char v)//在邻接矩阵表示的图G上插入顶点v
{
  if(G.vexnum+1)>MAX_VERTEX_NUM return INFEASIBLE;
  G.vexs[++G.vexnum]=v;
  return OK;
}//Insert_Vex
Status Insert_Arc(MGraph &G，char v，char w)//在邻接矩阵表示的图G上插入边(v，w)
{
  if((i=LocateVex(G，v))<0) return ERROR;
  if((j=LocateVex(G，w))<0) return ERROR;
  if(i==j) return ERROR;
  if(!G.arcs[i][j].adj)
  {
    G.arcs[i][j].adj=1;
    G.arcnum++;
  }
  return OK;
}//Insert_Arc
Status Delete_Vex(MGraph &G，char v)//在邻接矩阵表示的图G上删除顶点v
{
  n=G.vexnum;
  if((m=LocateVex(G，v))<0) return ERROR;
  G.vexs[m]<->G.vexs[n]; //将待删除顶点交换到最后一个顶点
  for(i=0;i<n;i++)
  {
    G.arcs[i][m]=G.arcs[i][n];
    G.arcs[m][i]=G.arcs[n][i]; //将边的关系随之交换
  }
  G.arcs[m][m].adj=0;
  G.vexnum--;
  return OK;
}//Delete_Vex
Status Delete_Arc(MGraph &G，char v，char w)//在邻接矩阵表示的图G上删除边(v，w)
```

```
{
  if((i=LocateVex(G, v))<0) return ERROR;
  if((j=LocateVex(G, w))<0) return ERROR;
  if(G.arcs[i][j].adj)
  {
    G.arcs[i][j].adj=0;
    G.arcnum--;
  }
  return OK;
}//Delete_Arc
```

14. 试对邻接表存储结构重做13题。

【分析】本题只给出Insert_Arc算法，其余算法请自行写出。

【解答】算法如下：

```
Status Insert_Arc(ALGraph &G, char v, char w)//在邻接表表示的图G上插入边(v, w)
{
  if((i=LocateVex(G, v))<0) return ERROR;
  if((j=LocateVex(G, w))<0) return ERROR;
  p=(ArcNode*)malloc(sizeof(ArcNode));
  p->adjvex=j;p->nextarc=NULL;
  if(!G.vertices[i].firstarc) G.vertices[i].firstarc=p;
  else
  {
    for(q=G.vertices[i].firstarc;q->q->nextarc=null; q=q->nextarc)
      if(q->adjvex==j) return ERROR; //边已经存在
    q->nextarc=p;
  }
  G.arcnum++;
  return OK;
}//Insert_Arc
```

15. 试对十字链表存储结构重做13题。

【分析】本题只给出较为复杂的Delete_Vex算法，其余算法请自行写出。

【解答】算法如下：

```
Status Delete_Vex(OLGraph &G, char v)//在十字链表表示的图G上删除顶点v
{
  if((m=LocateVex(G, v))<0) return ERROR;
  n=G.vexnum;
  for(i=0;i<n;i++) //删除所有以v为头的边
  {
    if(G.xlist[i].firstin->tailvex==m) //如果待删除的边是头链上的第一个结点
    {
      q=G.xlist[i].firstin;
      G.xlist[i].firstin=q->hlink;
      free(q);G.arcnum--;
    }
    else //否则
```

```
    {
      for(p=G.xlist[i].firstin;p&&p->hlink->tailvex!=m;p=p->hlink);
      if(p)
      {
        q=p->hlink;
        p->hlink=q->hlink;
        free(q);G.arcnum--;
      }
    }//else
  }//for
  for(i=0;i<n;i++) //删除所有以v为尾的边
  {
    if(G.xlist[i].firstout->headvex==m) //如果待删除的边是尾链上的第一个结点
    {
      q=G.xlist[i].firstout;
      G.xlist[i].firstout=q->tlink;
      free(q);G.arcnum--;
    }
    else //否则
    {
      for(p=G.xlist[i].firstout;p&&p->tlink->headvex!=m;p=p->tlink);
      if(p)
      {
        q=p->tlink;
        p->tlink=q->tlink;
        free(q);G.arcnum--;
      }
    }//else
  }//for
  for(i=m;i<n;i++) //顺次用结点m之后的顶点取代前一个顶点
  {
    G.xlist[i]=G.xlist[i+1]; //修改表头向量
    for(p=G.xlist[i].firstin;p;p=p->hlink)
      p->headvex--;
    for(p=G.xlist[i].firstout;p;p=p->tlink)
      p->tailvex--; //修改各链中的顶点序号
  }
  G.vexnum--;
  return OK;
}//Delete_Vex
```

16. 试对邻接多重表存储结构重做13题。

【分析】本题只给出Delete_Arc算法，其余算法请自行写出。

【解答】算法如下：

```
Status Delete_Arc(AMLGraph &G, char v, char w)//在邻接多重表表示的图G上删除边(v, w)
{
  if((i=LocateVex(G, v))<0) return ERROR;
  if((j=LocateVex(G, w))<0) return ERROR;
  if(G.adjmulist[i].firstedge->jvex==j)
```

```
    G.adjmulist[i].firstedge=G.adjmulist[i].firstedge->ilink;
  else
  {
    for(p=G.adjmulist[i].firstedge;p&&p->ilink->jvex!=j;p=p->ilink);
    if (!p) return ERROR; //未找到
    p->ilink=p->ilink->ilink;
  } //在i链表中删除该边
  if(G.adjmulist[j].firstedge->ivex==i)
    G.adjmulist[j].firstedge=G.adjmulist[j].firstedge->jlink;
  else
  {
    for(p=G.adjmulist[j].firstedge;p&&p->jlink->ivex!=i;p=p->jlink);
    if (!p) return ERROR; //未找到
    q=p->jlink;
    p->jlink=q->jlink;
    free(q);
  } //在i链表中删除该边
  G.arcnum--;
  return OK;
}//Delete_Arc
```

17. 编写算法，由依次输入的顶点数目、边的数目、各顶点的信息和各条边的信息建立无向图的邻接多重表。

【解答】算法如下：

```
Status Build_AdjMulist(AMLGraph &G)//输入有向图的顶点数，边数，顶点信息和边的信息建立邻接
                                   //多重表
{
  InitAMLGraph(G);
  scanf("%d", &v);
  if(v<0) return ERROR; //顶点数不能为负
  G.vexnum=v;
  scanf(%d", &a);
  if(a<0) return ERROR; //边数不能为负
  G.arcnum=a;
  for(m=0;m<v;m++)
    G.adjmulist[m].data=getchar(); //输入各顶点的符号
  for(m=1;m<=a;m++)
  {
    t=getchar();h=getchar(); //t为弧尾，h为弧头
    if((i=LocateVex(G, t))<0) return ERROR;
    if((j=LocateVex(G, h))<0) return ERROR; //顶点未找到
    p=(EBox*)malloc(sizeof(EBox));
    p->ivex=i;p->jvex=j;
    p->ilink=NULL;p->jlink=NULL; //边结点赋初值
    if(!G.adjmulist[i].firstedge) G.adjmulist[i].firstedge=p;
    else
    {
      q=G.adjmulist[i].firstedge;
```

```
      while(q)
      {
        r=q;
        if(q->ivex==i) q=q->ilink;
        else q=q->jlink;
      }
      if(r->ivex==i) r->ilink=p;//注意i值既可能出现在边结点的ivex域中，
      else r->jlink=p; //又可能出现在边结点的jvex域中
    }//else //插入i链表尾部
    if(!G.adjmulist[j].firstedge) G.adjmulist[j].firstedge=p;
    else
    {
      q=G.adjmulist[i].firstedge;
      while(q)
      {
        r=q;
        if(q->jvex==j) q=q->jlink;
        else q=q->ilnk;
      }
      if(r->jvex==j) r->jlink=p;
      else r->ilink=p;
    }//else //插入j链表尾部
  }//for
  return OK;
}//Build_AdjList
```

18．下面的算法段可以测定图$G=(V, E)$是否可传递：

```
trans=TRUE;
for (V中的每个x)
    for (N(x)中的每个y)
        for (N(y)中不等于x的每个z)
            if (z不在N(x)中)  trans=FALSE;
```

其中N(x)表示x邻接到的所有顶点的集合。试以邻接矩阵存储结构实现判定一个图的可传递性的算法，并通过n=| V |， m=| E | 和d=结点度数的均值，估计执行时间。

【解答】算法如下：

```
int Pass_MGraph(MGraph G)//判断一个邻接矩阵存储的有向图是不是可传递的，是则返回1，
                         //否则返回0
{
  for(x=0;x<G.vexnum;x++)
    for(y=0;y<G.vexnum;y++)
      if(G.arcs[x][y])
      {
        for(z=0;z<G.vexnum;z++)
          if(z!=x&&G.arcs[y][z]&&!G.arcs[x][z]) return 0;//图不可传递的条件
      }//if
  return 1;
}//Pass_MGraph
```

本算法的时间复杂度大概是$O(n^2*d)$。

19．试对邻接表存储结构重做18题。

【解答】算法如下：

```
int Pass_ALGraph(ALGraph G)//判断一个邻接表存储的有向图是不是可传递的，是则返回1，
                           //否则返回0
{
  for(x=0;x<G.vexnum;x++)
    for(p=G.vertices[x].firstarc;p;p=p->nextarc)
    {
      y=p->adjvex;
      for(q=G.vertices[y].firstarc;q;q=q->nextarc)
      {
        z=q->adjvex;
        if(z!=x&&!is_adj(G, x, z)) return 0;
      }//for
    }//for
}//Pass_ALGraph
int is_adj(ALGraph G, int m, int n)//判断有向图G中是否存在边(m, n)，是则返回1，否则返回
0
{
  for(p=G.vertices[m].firstarc;p;p=p->nextarc)
    if(p->adjvex==n) return 1;
  return 0;
}//is_adj
```

20．试基于图的深度优先搜索策略写一算法，判别以邻接表方式存储的有向图中是否存在由顶点v_i到顶点v_j的路径（$i\neq j$）。注意：算法中涉及的图的基本操作必须在此存储结构上实现。

【分析】深度优先搜索与递归有着本质的联系，在递归程序中，调用结构能用图表示为一棵有根树，它的每个顶点表示关于这个程序的递归调用，执行这些调用的次序对应于这棵树的深度优先遍历。

【解答】算法如下：

```
int visited[MAXSIZE]; //指示顶点是否在当前路径上
int exist_path_DFS(ALGraph G, int i, int j)//深度优先判断有向图G中顶点i到顶点j是否有路
径，
                                    //是则返回1，否则返回0
{
  if(i==j) return 1; //i就是j
  else
  {
    visited[i]=1;
    for(p=G.vertices[i].firstarc;p;p=p->nextarc)
    {
      k=p->adjvex;
      if(!visited[k]&&exist_path(k, j)) return 1;//i下游的顶点到j有路径
```

```
    }//for
  }//else
}//exist_path_DFS
```

21．同20题要求。试基于图的广度优先搜索策略写一算法。

【解答】算法如下：

```
int exist_path_BFS(ALGraph G, int i, int j)//广度优先判断有向图G中顶点i到顶点j是否有路径，是则
                                           //返回1，否则返回0
{
  int visited[MAXSIZE];
  InitQueue(Q);
  EnQueue(Q, i);
  while(!QueueEmpty(Q))
  {
    DeQueue(Q, u);
    visited[u]=1;
    for(p=G.vertices[i].firstarc;p;p=p->nextarc)
    {
      k=p->adjvex;
      if(k==j) return 1;
      if(!visited[k]) EnQueue(Q, k);
    }//for
  }//while
  return 0;
}//exist_path_BFS
```

22．试利用栈的基本操作编写按深度优先搜索策略遍历一个强连通图的非递归形式的算法。算法中不规定具体的存储结构，而将图Graph看成是一种抽象的数据类型。

【解答】算法如下：

```
void STraverse_Nonrecursive(Graph G)//非递归遍历强连通图G
{
  int visited[MAXSIZE];
  InitStack(S);
  Push(S, GetVex(S, 1)); //将第一个顶点入栈
  visit(1);
  visited =1;
  while(!StackEmpty(S))
  {
    while(Gettop(S, i)&&i)
    {
      j=FirstAdjVex(G, i);
      if(j&&!visited[j])
      {
        visit(j);
        visited[j]=1;
        Push(S, j); //向左走到尽头
      }
```

```
    }//while
    if(!StackEmpty(S))
    {
      Pop(S, j);
      Gettop(S, i);
      k=NextAdjVex(G, i, j); //向右走一步
      if(k&&!visited[k])
      {
        visit(k);
        visited[k]=1;
        Push(S, k);
      }
    }//if
  }//while
}//Straverse_Nonrecursive
```

23.假设对有向图中n个顶点进行自然编号，并以三个数组s[1..max]，fst[1..n]和lst[1..n]表示之。其中数组s存放每个顶点的后继顶点的信息，第i个顶点的后继顶点存放在s中下标从fst[i]起到lst[i]的分量中（i=1，2，…，n）。若fst[i]>lst[i]，则第i个顶点无后继顶点。试编写判别该有向图中是否存在回路的算法。

【解答】图中存在经过V_i的回路的充分必要条件是$a_{i,j}^{(n)} \neq 0$（图论知识：若一个图中有回路，则必存在长度不超过n的（简单）回路）。这种方法在理论上显得简洁，但算法的时间复杂度高达$O(n^4)$，又不适用于本题中给出的存储结构，因而是不足取的。

必须看到，遍历一个图不过是按一种特定的方式搜索一个图：每个结点被访问且只访问一次。它能够解决的问题类型是这样的：寻找满足给定条件P的第一个（或一些，或所有）结点；或者判别从某个给定结点到达其他结点的可达性，其中，P是一个命题，例如“从给定结点到达它的最短路径为4”。也可以就是“真”。然而，比上述问题类大得多的一类问题，即人工智能中所讨论的问题求解的一般提法是：在给定图（甚至不限于有穷图）中寻找从给定初始结点到满足给定条件P的第一个（或一些或所有）满足条件Q的简单路径。由路径的定义（相邻顶点之间存在边或弧的顶点序列）可知，“求”路径即为在遍历的过程中记录属于“当前所求路径”的顶点。此时，常用的遍历算法就不适用了，原因是没有记录路径（由traver算法所得顶点序列不一定是“路径”），特别是没有行遍所有简单路径。下面给出按深度优先策略进行“路径”遍历的算法的基本框架：

```
void PathDFS (Graph &G; vexindex v)
{// 在G中找以当前路径为前缀的、到达满足条件P的结点的、所有满足条件Q的简单路径，并印出它们。V为当前考察的顶点
     if (!OnCurrentPath [v] )
     {
          OnCurrentPath [v]=TRUE;
          EnCurrentPath (v, CurrentPath);// 将当前访问的结点v置为当前路径上的一个新的结点。
          // 初始时，数组OnCurrentPath [vexindex]为全“假”。
          // EnCurrentPath表示往路径中添加结点。
          if (P (v) &&Q(CurrentPath )) print (CurrentPath);
```

```
        else {
                w=FirstAdjVex (G, v);
                while (w) { PathDFS (G, w);  w=NextAdjVex (G, v, w); }
        }//visit(v)
        OnCurrentPath [v]=FALSE;
        DeCurrentPath (v, Current);      //把当前结点v从当前路径上删除
    }
} //pathdfs
```

如果只要求作出一条路径，问题要简单得多，因为递归调用的各层局部参量（注意，在某一层中不能存取它们)恰好是当前路径的结点序列!所以只要设全程布尔量successful（初值为假），将“print (CurrentPath)”改为Successful=TRUE；以（！Successful）为条件执行visit之后的两条语句，并且在后一条（即while）之后插入“if (Successful) printf (v)”。打印操作的位置不变，而只设布尔量也是可以的，但很多问题较简单，不涉及条件Q。这时数据结构CurrentPath以及涉及它的操作就可被求精掉了，使程序大大简化。这个模式尚不适于最优解问题，但只要稍加改变就成为求最优路径问题的模式，如“最短路径”。

回路不是简单路径。求回路的模式可以通过对上述模式稍加修改而得到。

有兴趣的读者可以继续研究路径遍历的广度优先算法，参阅《人工智能原理》（N.J.Nilson著，石纯一译）。

24. 对有向图中顶点适当地编号，可使其邻接矩阵为下三角形且主对角线为全零的充要条件是：该有向图不含回路。请编写一算法对无环有向图的顶点重新编号，使其邻接矩阵变为下三角形，并输出新旧编号对照表。

【分析】只需按逆序对顶点编号，就可以使邻接矩阵成为下三角矩阵。

【解答】算法如下：

```
Status TopoNo(ALGraph G)//按照题目要求顺序重排有向图中的顶点
{
  int new[MAXSIZE], indegree[MAXSIZE]; //储存结点的新序号
  n=G.vexnum;
  FindInDegree(G, indegree);
  InitStack(S);
  for(i=1;i<G.vexnum;i++)
    if(!indegree[i]) Push(S, i); //零入度结点入栈
  count=0;
  while(!StackEmpty(S))
  {
    Pop(S, i);
    new[i]=n--; //记录结点的逆序序号
    count++;
    for(p=G.vertices[i].firstarc;p;p=p->nextarc)
    {
      k=p->adjvex;
      if(!(--indegree[k])) Push(S, k);
    }//for
```

```
  }//while
  if(count<G.vexnum) return ERROR; //图中存在环
  for(i=1;i<=n;i++) printf("Old No:%d New No:%d\n", i, new[i])
  return OK;
}//TopoNo
```

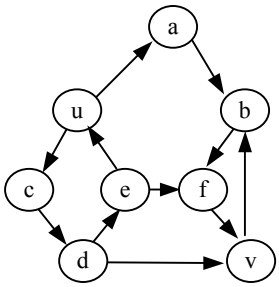
25. 采用链接表存储结构，编写一个判别无向图中任意给定的两个顶点之间是否存在一条长度为k的简单路径的算法。

【分析】应采用（限制深度的）深度优先策略，必须使用路径遍历算法。读者可以以图{（v_1，v_2），（v_2，v_3），（v_3，v_4）}k=2，为例进行验证：判断v_1到v_4之间是否存在长度为2的简单路径。

【解答】算法如下：

```
int visited[MAXSIZE];
int exist_path_len(ALGraph G, int i, int j, int k)//判断邻接表方式存储的有向图G的顶点i到j是否存在
                                                  //长度为k的简单路径
{
  if(i==j&&k==0) return 1; //找到了一条路径，且长度符合要求
  else if(k>0)
     {
        visited[i]=1;
        for(p=G.vertices[i].firstarc;p;p=p->nextarc)
        {
            l=p->adjvex;
            if(!visited[l])
               if(exist_path_len(G, l, j, k-1)) return 1; //剩余路径长度减一
        }//for
        visited[i]=0; //本题允许曾经被访问过的结点出现在另一条路径中
     }//else
  return 0; //没找到
}//exist_path_len
```


26. 已知有向图和图中两个顶点u和v，试编写算法求有向图中从u到v的所有简单路径，并以下图为例手工执行你的算法，画出相应的搜索过程图。



【分析】最典型的路径遍历问题，本题实际上是一般化的走迷宫问题，只要将那里迷宫中的通道方格视为结点，“相邻”关系视为无向边。数据结构“CurrentPath”必须维护。

【解答】算法如下：

```
int path[MAXSIZE], visited[MAXSIZE]; //暂存遍历过程中的路径
int Find_All_Path(ALGraph G, int u, int v, int k)//求有向图G中顶点u到v之间的所有简单路径，k表
                                                  //示当前路径长度
{
  path[k]=u; //加入当前路径中
  visited[u]=1;
  if(u==v) //找到了一条简单路径
  {
    printf("Found one path!\n");
    for(i=0;path[i];i++) printf("%d", path[i]); //打印输出
  }
  else
    for(p=G.vertices[u].firstarc;p;p=p->nextarc)
    {
       l=p->adjvex;
       if(!visited[l]) Find_All_Path(G, l, v, k+1); //继续寻找
    }
  visited[u]=0;
  path[k]=0; //回溯
}//Find_All_Path
main()
{
  ...
  Find_All_Path(G, u, v, 0); //在主函数中初次调用，k值应为0
  ...
}//main
```

27. 试写一个算法，在以邻接矩阵方式存储的有向图G中求顶点i到顶点j的不含回路的长度为k的路径数。

【分析】本题关键是设置visit数组，并采用分治法写递归程序。

【解答】算法如下：

```
int GetPathNum_Len(ALGraph G, int i, int j, int len)//求邻接表方式存储的有向图G的顶点i到j之间长
                                                    //度为len的简单路径条数
{
  if(i==j&&len==0) return 1; //找到了一条路径，且长度符合要求
  else if(len>0)
     {
         sum=0; //sum表示通过本结点的路径数
         visited[i]=1;
         for(p=G.vertices[i].firstarc;p;p=p->nextarc)
         {
             l=p->adjvex;
             if(!visited[l])
                 sum+=GetPathNum_Len(G, l, j, len-1)//剩余路径长度减一
         }//for
         visited[i]=0; //本题允许曾经被访问过的结点出现在另一条路径中
     }//else
```

```
  return sum;
}//GetPathNum_Len
```

28．试写一个求有向图*G*中所有简单回路的算法。

【分析】这个算法的思想是，在遍历中暂存当前路径，当遇到一个结点已经在路径之中时就表明存在一条回路；扫描路径向量path可以获得这条回路上的所有结点。把结点序列（例如，142857）存入thiscycle中；由于这种算法中，一条回路会被发现好几次，所以必须先判断该回路是否已经在cycles中被记录过，如果没有才能存入cycles的一个行向量中。把cycles的每一个行向量取出来与之比较。由于一条回路可能有多种存储顺序，比如142857等同于285714和571428，所以还要调整行向量的次序，由于142857和857142是同一条回路，因此不予存储。

【解答】算法如下：

```
int visited[MAXSIZE];
int path[MAXSIZE]; //暂存当前路径
int cycles[MAXSIZE][MAXSIZE]; //储存发现的回路所包含的结点
int thiscycle[MAXSIZE]; //储存当前发现的一个回路
int cycount=0; //已发现的回路个数

void GetAllCycle(ALGraph G)//求有向图中所有的简单回路
{
  for(v=0;v<G.vexnum;v++) visited[v]=0;
  for(v=0;v<G.vexnum;v++)
    if(!visited[v]) DFS(G, v, 0); //深度优先遍历
}//DFSTraverse
void DFS(ALGraph G, int v, int k)//k表示当前结点在路径上的序号
{
  visited[v]=1;
  path[k]=v; //记录当前路径
  for(p=G.vertices[v].firstarc;p;p=p->nextarc)
  {
    w=p->adjvex;
    if(!visited[w]) DFS(G, w, k+1);
    else //发现了一条回路
    {
       for(i=0;path[i]!=w;i++); //找到回路的起点
       for(j=0;path[i+j];i++) thiscycle[j]=path[i+j];//把回路复制下来
       if(!exist_cycle()) cycles[cycount++]=thiscycle;//如果该回路尚未被记录过，就添加到记录中
       for(i=0;i<G.vexnum;i++) thiscycle[i]=0; //清空目前回路数组
    }//else
  }//for
  path[k]=0;
  visited[k]=0; //注意只有当前路径上的结点visited为真,因此一旦遍历中发现当前结点visited为真,
            //即表示发现了一条回路
}//DFS
int exist_cycle()//判断thiscycle数组中记录的回路在cycles的记录中是否已经存在
```

```
{
  int temp[MAXSIZE];
  for(i=0;i<cycount;i++) //判断已有的回路与thiscycle是否相同
  { //也就是，所有结点和它们的顺序都相同
      j=0;c=thiscycle&#0;; //例如，142857和857142是相同的回路
      for(k=0;cycles[i][k]!=c&&cycles[i][k]!=0;k++);//在cycles的一个行向量中寻找等于
thiscycle第一
                                                  //个结点的元素
      if(cycles[i][k]) //有与之相同的一个元素
      {
          for(m=0;cycles[i][k+m];m++)
          temp[m]=cycles[i][k+m];
          for(n=0;n<k;n++, m++)
          temp[m]=cycles[i][n]; //调整cycles中的当前记录的循环相位并放入temp数组中
          if(!StrCompare(temp, thiscycle)) //与thiscycle比较
             return 1; //完全相等
          for(m=0;m<G.vexnum;m++) temp[m]=0; //清空这个数组
      }
  }//for
  return 0; //所有现存回路都不与thiscycle完全相等
}//exist_cycle
```

29．试完成求有向图的强连通分量的算法，并分析算法的时间复杂度。

【解答】算法如下：

```
int visited[MAXSIZE];
int finished[MAXSIZE];
int count; //count在第一次深度优先遍历中用于指示finished数组的填充位置
void Get_SGraph(OLGraph G)//求十字链表结构储存的有向图G的强连通分量
{
  count=0;
  for(v=0;v<G.vexnum;v++) visited[v]=0;
  for(v=0;v<G.vexnum;v++) //第一次深度优先遍历建立finished数组
    if(!visited[v]) DFS1(G, v);
  for(v=0;v<G.vexnum;v++) visited[v]=0; //清空visited数组
  for(i=G.vexnum-1;i>=0;i--) //第二次逆向的深度优先遍历
  {
    v=finished(i);
    if(!visited[v])
    {
      printf("\n"); //不同的强连通分量在不同的行输出
      DFS2(G, v);
    }
  }//for
}//Get_SGraph
void DFS1(OLGraph G, int v)//第一次深度优先遍历的算法
{
  visited[v]=1;
  for(p=G.xlist[v].firstout;p;p=p->tlink)
  {
```

```
    w=p->headvex;
    if(!visited[w]) DFS1(G, w);
  }//for
  finished[++count]=v; //在第一次遍历中建立finished数组
}//DFS1
void DFS2(OLGraph G, int v)//第二次逆向的深度优先遍历的算法
{
  visited[v]=1;
  printf("%d", v); //在第二次遍历中输出结点序号
  for(p=G.xlist[v].firstin;p;p=p->hlink)
  {
    w=p->tailvex;
    if(!visited[w]) DFS2(G, w);
  }//for
}//DFS2
```

求有向图的强连通分量的算法的时间复杂度和深度优先遍历相同，也为$O(n+e)$。

30．试修改普里姆算法，使之能在邻接表存储结构上实现求图的最小生成森林，并分析其时间复杂度（森林的存储结构为孩子-兄弟链表）。

【分析】在Prim算法基础上添加了非连通图支持和孩子-兄弟链表构建模块而得到的，其时间复杂度为$O(n^2)$。

【解答】算法如下：

```
void Forest_Prim(ALGraph G, int k, CSTree &T)//从顶点k出发，构造邻接表结构的有向图G的最小生
                                              //成森林T，用孩子兄弟链表存储
{
  for(j=0;j<G.vexnum;j++) //以下在Prim算法基础上稍作改动
    if(j!=k)
    {
      closedge[j]={k, Max_int};
      for(p=G.vertices[j].firstarc;p;p=p->nextarc)
        if(p->adjvex==k) closedge[j].lowcost=p->cost;
    }//if
  closedge[k].lowcost=0;
  for(i=1;i<G.vexnum;i++)
  {
    k=minimum(closedge);
    if (closedge[k].lowcost<Max_int)
    {
      Addto_Forest(T, closedge[k].adjvex, k); //把这条边加入生成森林中
      closedge[k].lowcost=0;
      for(p=G.vertices[k].firstarc;p;p=p->nextarc)
        if(p->cost<closedge[p->adjvex].lowcost)
          closedge[p->adjvex]={k, p->cost};
    }//if
    else Forest_Prim(G, k); //对另外一个连通分量执行算法
  }//for
```

```
}//Forest_Prim
void Addto_Forest(CSTree &T, int i, int j)//把边(i, j)添加到孩子兄弟链表表示的树T中
{
  p=Locate(T, i); //找到结点i对应的指针p，过程略
  q=(CSTNode*)malloc(sizeof(CSTNode));
  q->data=j;
  if(!p) //起始顶点不属于森林中已有的任何一棵树
  {
    p=(CSTNode*)malloc(sizeof(CSTNode));
    p->data=i;
    for(r=T;r->nextsib;r=r->nextsib);
    r->nextsib=p;
    p->firstchild=q;
  } //作为新树插入到最右侧
  else if(!p->firstchild) //双亲还没有孩子
         p->firstchild=q; //作为双亲的第一个孩子
     else //双亲已经有了孩子
     {
        for(r=p->firstchild;r->nextsib;r=r->nextsib);
           r->nextsib=q; //作为双亲最后一个孩子的兄弟
     }
}//Addto_Forest
main()
{
  ...
  T=(CSTNode*)malloc(sizeof(CSTNode)); //建立树根
  T->data=1;
  Forest_Prim(G, 1, T);
  ...
}//main
```

这个算法是在Prim算法的基础上添加了非连通图支持和孩子兄弟链表构建模块而得到的，其时间复杂度为$O(n^2)$。

31．已知无向图的边集存放在某个类型为EdgeSetType的数据结构EdgeSet中（没有端点相重的环边），并在此结构上已定义两种基本运算：

（1）函数GetMinEdge(EdgeSet，u，v)：若EdgeSet非空，则必存在最小边，变参u和v分别含最小边上两顶点，并返回true；否则返回false；

（2）过程DelMinEdge(EdgeSet，u，v)：从EdgeSet中删除依附于顶点u和v的最小边。

试在上述结构上实现求最小生成树（以孩子-兄弟链表表示）的克鲁斯卡尔算法。

【分析】设计一个数据结构及其之上的操作表示等价类。运算包括初始化、判等价、合并两个类等。数据结构的选取可以组织为静态循环链表。要先考虑怎样把两个循环链表合并为一个循环链表。数据结构的选取原则：使以上所述的操作易于高效地实现。本算法使用一维结构体变量数组来表示等价类，每个连通分量所包含的所有结点属于一个等价类，在这个结构上实现了初始化、判断元素是否等价（两个结点是否属于同个连通分量）、合并等价类（连通分量）的操作。

【解答】算法如下：

```
typedef struct {
                int vex; //结点序号
                int ecno; //结点所属的连通分量号
            } VexInfo;
VexInfo vexs[MAXSIZE]; //记录结点所属连通分量号的数组
void Init_VexInfo(VexInfo &vexs[ ], int vexnum)//初始化
{
  for(i=0;i<vexnum;i++)
    vexs[i]={i, i}; //初始状态:每一个结点都属于不同的连通分量
}//Init_VexInfo
int is_ec(VexInfo vexs[ ], int i, int j)//判断顶点i和顶点j是否属于同一个连通分量
{
  if(vexs[i].ecno==vexs[j].ecno) return 1;
  else return 0;
}//is_ec
void merge_ec(VexInfo &vexs[ ], int ec1, int ec2)//合并连通分量ec1和ec2
{
  for(i=0;vexs[i].vex;i++)
    if(vexs[i].ecno==ec2) vexs[i].ecno==ec1;
}//merge_ec
void MinSpanTree_Kruscal(Graph G, EdgeSetType &EdgeSet, CSTree &T)//求图的最小生成树的
克鲁
                                                                  //斯卡尔算法
{
  Init_VexInfo(vexs, G.vexnum);
  ecnum=G.vexnum; //连通分量个数
  while(ecnum>1)
  {
    GetMinEdge(EdgeSet, u, v); //选出最短边
    if(!is_ec(vexs, u, v)) //u和v属于不同连通分量
    {
      Addto_CSTree(T, u, v); //加入到生成树中
      merge_ec(vexs, vexs[u].ecno, vexs[v].ecno); //合并连通分量
      ecnum--;
    }
    DelMinEdge(EdgeSet, u, v); //从边集中删除
  }//while
}//MinSpanTree_Kruscal
void Addto_CSTree(CSTree &T, int i, int j)//把边(i, j)添加到孩子兄弟链表表示的树T中
{
  p=Locate(T, i); //找到结点i对应的指针p，过程略
  q=(CSTNode*)malloc(sizeof(CSTNode));
  q->data=j;
  if(!p->firstchild) //双亲还没有孩子
    p->firstchild=q; //作为双亲的第一个孩子
  else //双亲已经有了孩子
  {
    for(r=p->firstchild;r->nextsib;r=r->nextsib);
```

```
    r->nextsib=q; //作为双亲最后一个孩子的兄弟
  }
}//Addto_CSTree
```

32. 试编写一个算法，给有向无环图*G*中每个顶点赋以一个整数序号，并满足以下条件：若从顶点*i*至顶点*j*有一条弧，则应使*i*<*j*。

【解答】算法如下：

```
Status TopoSeq(ALGraph G, int new[ ])//按照题目要求给有向无环图的结点重新编号，并存入数组
                                     //new中
{
  int indegree[MAXSIZE]; //本算法就是拓扑排序
  FindIndegree(G, indegree);
  Initstack(S);
  for(i=0;i<G.vexnum;i++)
    if(!indegree[i]) Push(S, i);
  count=0;
  while(!stackempty(S))
  {
    Pop(S, i);new[i]=++count; //把拓扑顺序存入数组的对应分量中
    for(p=G.vertices[i].firstarc;p;p=p->nextarc)
    {
      k=p->adjvex;
      if(!(--indegree[k])) Push(S, k);
    }
  }//while
  if(count<G.vexnum) return ERROR;
  return OK;
}//TopoSeq
```

33. 若在*DAG*图中存在一个顶点*r*，在*r*和图中所有其他顶点之间均存在由*r*出发的有向路径，则称该*DAG*图有根。试编写求*DAG*图的根的算法。

【分析】考察*DAG*图的逆邻接表可知，从每个结点出发的深度优先路径遍历（不是结点遍历）中所有回溯点相同的充要条件为*DAG*有根。但是，路径遍历时间复杂度较高，应考虑按结点遍历时，*DAG*有根的充要条件是什么。

【解答】算法如下：

```
int visited[MAXSIZE];
void Get_Root(ALGraph G)//求有向无环图的根，如果有的话
{
  for(v=0;v<G.vexnum;v++)
  {
    for(w=0;w<G.vexnum;w++) visited[w]=0;//每次都要将访问数组清零
    DFS(G, v); //从顶点v出发进行深度优先遍历
    for(flag=1, w=0;w<G.vexnum;w++)
      if(!visited[w]) flag=0; //如果v是根，则深度优先遍历可以访问到所有结点
    if(flag) printf("Found a root vertex:%d\n", v);
  }//for
```

```
}//Get_Root，这个算法要求图中不能有环，否则会发生误判
void DFS(ALGraph G, int v)
{
  visited[v]=1;
  for(p=G.vertices[v].firstarc;p;p=p->nextarc)
  {
    w=p->adjvex;
    if(!visited[w]) DFS(G, w);
  }
}//DFS
```

34．在图的邻接表存储结构中，为每个顶点增加一个MPL域。试写一算法，求有向无环图*G*的每个顶点出发的最长路径的长度，并存入其MPL域。请给出算法的时间复杂度。

【解答】算法如下：

```
void Fill_MPL(ALGraph &G)//为有向无环图G添加MPL域
{
  FindIndegree(G, indegree);
  for(i=0;i<G.vexnum;i++)
    if(!indegree[i]) Get_MPL(G, i);//从每一个零入度顶点出发构建MPL域
}//Fill_MPL
int Get_MPL(ALGraph &G, int i)//从一个顶点出发构建MPL域并返回其MPL值
{
  if(!G.vertices[i].firstarc)
  {
    G.vertices[i].MPL=0;
    return 0; //零出度顶点
  }
  else
  {
    max=0;
    for(p=G.vertices[i].firstarc;p;p=p->nextarc)
    {
      j=p->adjvex;
      if(G.vertices[j].MPL==0) k=Get_MPL(G, j);
      if(k>max) max=k; //求其直接后继顶点MPL的最大者
    }
    G.vertices[i]=max+1;//再加一，就是当前顶点的MPL
    return max+1;
  }//else
}//Get_MPL
```

该算法的时间复杂度为$O(n^2)$。

35．试设计一个求有向无环图中最长路径的算法，并估计其时间复杂度。

【分析】考察以任一个特定结点为起点的最长路径，若无所邻接到的结点，则它的长度为0；否则为其诸邻接点的最长路径中的最大值。最后思考：上述思想方法在非*DAG*图中会怎样。

【解答】算法如下：

```
int maxlen, path[MAXSIZE]; //数组path用于存储当前路径
int mlp[MAXSIZE]; //数组mlp用于存储已发现的最长路径
void Get_Longest_Path(ALGraph G)//求一个有向无环图中最长的路径
{
  maxlen=0;
  FindIndegree(G, indegree);
  for(i=0;i<G.vexnum;i++)
  {
    for(j=0;j<G.vexnum;j++) visited[j]=0;
    if(!indegree[i]) DFS(G, i, 0);//从每一个零入度结点开始深度优先遍历
  }
  printf("Longest Path:");
  for(i=0;mlp[i];i++) printf("%d", mlp[i]); //输出最长路径
}//Get_Longest_Path
void DFS(ALGraph G, int i, int len)
{
  visited[i]=1;
  path[len]=i;
  if(len>maxlen&&!G.vertices[i].firstarc) //新的最长路径
  {
    for(j=0;j<=len;j++) mlp[j]=path[j]; //保存下来
    maxlen=len;
  }
  else
  {
    for(p=G.vertices[i].firstarc;p;p=p->nextarc)
    {
      j=p->adjvex;
      if(!visited[j]) DFS(G, j, len+1);
    }
  }//else
  path[i]=0;
  visited[i]=0;
}//DFS
```

该算法的时间复杂度为$O(n^3)$。

36. 一个四则运算算术表达式以有向无环图的邻接表方式存储，每个操作数原子都由单个字母表示。写一个算法输出其逆波兰表达式。

【解答】算法如下：

```
void NiBoLan_DAG(ALGraph G)//输出有向无环图形式表示的表达式的逆波兰式
{
  FindIndegree(G, indegree);
  for(i=0;i<G.vexnum;i++)
    if(!indegree[i]) r=i; //找到有向无环图的根
  PrintNiBoLan_DAG(G, i);
}//NiBoLan_DAG
void PrintNiBoLan_DAG(ALGraph G, int i)//打印输出以顶点i为根的表达式的逆波兰式
{
```

```
  c=G.vertices[i].data;
  if(!G.vertices[i].firstarc) printf("%c", c); //c是原子
  else //子表达式
  {
    p=G.vertices[i].firstarc;
    PrintNiBoLan_DAG(G, p->adjvex);
    PrintNiBoLan_DAG(G, p->nexarc->adjvex);
    printf("%c", c);
  }
}//PrintNiBoLan_DAG
```

37. 把存储结构改为二叉链表，重做36题。

【解答】算法如下：

```
void PrintNiBoLan_Bitree(Bitree T)//在二叉链表存储结构上重做上一题
{
  if(T->lchild) PrintNiBoLan_Bitree(T->lchild);
  if(T->rchild) PrintNiBoLan_Bitree(T->rchild);
  printf("%c", T->data);
}//PrintNiBoLan_Bitree
```

38. 若36题的运算符和操作数原子分别由字符和整数表示，请设计邻接表的结点类型，并且写一个表达式求值的算法。

【分析】本题中，邻接表的vertices向量的元素类型修改如下：

```
struct {
         enum tag{NUM, OPTR};
         union {
                 int value;
                 char optr;
               };
         ArcNode * firstarc;
       } Elemtype;
```

【解答】算法如下：

```
int Evaluate_DAG(ALGraph G)//给有向无环图表示的表达式求值
{
  FindIndegree(G, indegree);
  for(i=0;i<G.vexnum;i++)
    if(!indegree[i]) r=i; //找到有向无环图的根
  return Evaluate_imp(G, i);
}//NiBoLan_DAG
int Evaluate_imp(ALGraph G, int i)//求子表达式的值
{
  if(G.vertices[i].tag=NUM) return G.vertices[i].value;
  else
  {
    p=G.vertices[i].firstarc;
    v1=Evaluate_imp(G, p->adjvex);
```

```
    v2=Evaluate_imp(G, p->nextarc->adjvex);
    return calculate(v1, G.vertices[i].optr, v2);
  }
}//Evaluate_imp
```

39. 试编写利用深度优先遍历有向图实现求关键路径的算法。

【解答】算法如下：

```
void Critical_Path(ALGraph G)//利用深度优先遍历求网的关键路径
{
  FindIndegree(G, indegree);
  for(i=0;i<G.vexnum;i++)
    if(!indegree[i]) DFS1(G, i); //第一次深度优先遍历:建立ve
  for(i=0;i<G.vexnum;i++)
    if(!indegree[i]) DFS2(G, i); //第二次深度优先遍历:建立vl
  for(i=0;i<=G.vexnum;i++)
    if(vl[i]==ve[i]) printf("%d", i); //打印输出关键路径
}//Critical_Path
void DFS1(ALGraph G, int i)
{
  if(!indegree[i]) ve[i]=0;
  for(p=G.vertices[i].firstarc;p;p=p->nextarc)
  {
    dut=*p->info;
    if(ve[i]+dut>ve[p->adjvex])
      ve[p->adjvex]=ve[i]+dut;
    DFS1(G, p->adjvex);
  }
}//DFS1
void DFS2(ALGraph G, int i)
{
  if(!G.vertices[i].firstarc) vl[i]=ve[i];
  else
  {
    for(p=G.vertices[i].firstarc;p;p=p->nextarc)
    {
      DFS2(G, p->adjvex);
      dut=*p->info;
      if(vl[p->adjvex]-dut<vl[i])
        vl[i]=vl[p->adjvex]-dut;
    }
  }//else
}//DFS2
```

40. 以邻接表作存储结构实现求从源点到其余各顶点的最短路径的Dijkstra算法。

【分析】对Dijkstra算法中直接取任意边长度的语句作修改，由于在原算法中，每一次循环都是对尾相同的边进行处理，所以可以用遍历邻接表中的一条链来代替。

【解答】算法如下：

```
void ALGraph_DIJ(ALGraph G, int v0, Pathmatrix &P, ShortestPathTable &D)
                              //在邻接表存储结构上实现Dijkstra算法
{
  for(v=0;v<G.vexnum;v++)
    D[v]=INFINITY;
  for(p=G.vertices[v0].firstarc;p;p=p->nextarc)
    D[p->adjvex]=*p->info; //给D数组赋初值
  for(v=0;v<G.vexnum;v++)
  {
    final[v]=0;
    for(w=0;w<G.vexnum;w++) P[v][w]=0; //设空路径
    if(D[v]<INFINITY)
    {
      P[v][v0]=1;
      P[v][v]=1;
    }
  }//for
  D[v0]=0;final[v0]=1; //初始化
  for(i=1;i<G.vexnum;i++)
  {
    min=INFINITY;
    for(w=0;w<G.vexnum;w++)
      if(!final[w])
        if(D[w]<min) //尚未求出到该顶点的最短路径
        {
          v=w;
          min=D[w];
        }
    final[v]=1;
    for(p=G.vertices[v].firstarc;p;p=p->nextarc)
    {
      w=p->adjvex;
      if(!final[w]&&(min+(*p->info)<D[w])) //符合Dijkstra条件
      {
        D[w]=min+edgelen(G, v, w);
        P[w]=P[v];
        P[w][w]=1; //构造最短路径
      }
    }//for
  }//for
}//ALGraph_DIJ
```

7.4 考研真题分析

例题7-1 （清华大学1998年试题）

设无向图的顶点个数为n，则该无向图最多有_____条边。

A. n-1　　B. $n(n-1)/2$　　C. $n(n+1)/2$　　D. 0　　E. n^2

【解答】B

例题7-2 （中国科学院计算技术研究所1999年试题）

对于给定的n个元素，可以构造出的逻辑结构有_____、_____、_____和_____4种。

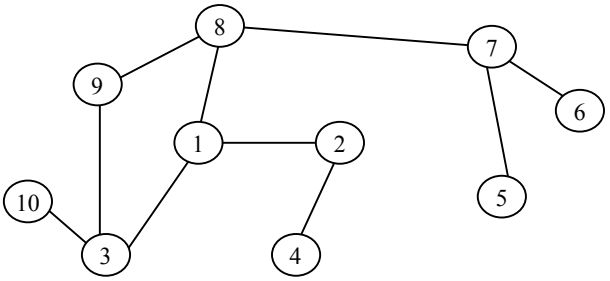
【解答】集合，线性结构，树形结构，图结构

例题7-3 （清华大学1998年试题）

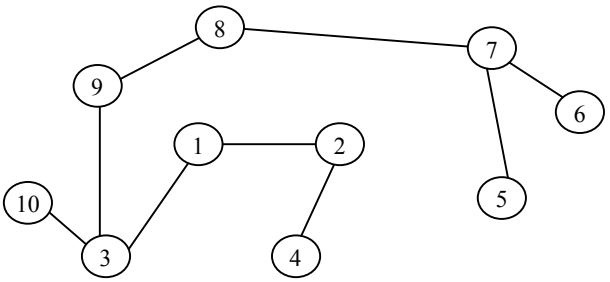
对下图所示的连通图，请画出：

① 以顶点1为根的深度优先生成树；

② 如果有关结点，请找出所有关结点。



【解答】① 以顶点1为根的深度优先生成树如下图所示。



② 图中的关结点是1、2、3、7、8。

例题7-4 （中国科学院计算技术研究所1999年试题）

Kruskal算法的时间复杂度为_____，它对_____图较为合适。

【解答】$O(e\log e)$，稀疏

例题7-5 （中国科学院计算技术研究所1999年试题）

自然树（即无环连通图）$T=(V, E)$的直径是树中所有点对点间最短路径长度的最大值，即T的直径定义为MAX $d(u, v)(u, v\in V)$，这里$d(u, v)$表示顶点u到顶点v的最短路径长度（路径长度为路径中包含的边数）。试写一算法求T的直径，并分析算法的时间复杂度（时

间复杂度越小得分越高）。

【分析】对于一般的图，求出图中每一对顶点之间的最短路径的时间复杂度为$O(n^3)$。在本题中，因为给出的是一个无环连通图，故有更有效的方法。主要思想是：对于一棵树来说，每对结点之间存在惟一的一条路径；整个树的直径必定是距离根最远的两个叶子结点之间的距离。可以利用深度优先遍历方法求出图的直径。

【解答】

```
FUNC find_diameter(g:Graph):integer;
{利用深度优先遍历方法求出根结点到每一个叶子结点的距离，maxlen1存放的是目前找到根到叶子
结点的最大值，maxlen2存放的是目前找到根到叶子结点的次大值，len记录深度优先遍历中到某一
叶子结点的距离，设图g的顶点编号为1到vtxnum，以顶点1为根生成深度优先树}
maxlen1:=0;
maxlen2:=0;
len:=0;
w:=firstadj(g, 1);{w为顶点1的邻接点}
WHILE w<>0 DO
BEGIN   {当邻接点存在时}
     len:=length(g, 1, w); {顶点1到顶点w的距离}
     dfs(g, 1, w, len);
     IF len>maxlen1
     THEN BEGIN
          maxlen2:=maxlen1;
          maxlen1:=len;
     END
     ELSE IF len>maxlen2
          maxlen2:=len;
          w:=nextadj(g, 1, w);
     END
     Return(maxlen1+maxlen2);
ENDF;{find_diameter}

PROC dfs (g:Graph;parent, child:vtxptr;VAR len:integer);
{dfs 返回时，len中存放的是从根到结点child所在的子树的叶子结点的最大距离}
     current_len:=len;
     maxlen:=len;
     v:=firstadj(g, child);
     WHILE v<>0 DO BEGIN
        IF v=parent CONTINUE;
        len:=len+length(g, child,  v);
        dfs(g, child, v, len);
        IF len>maxlen THEN
             maxlen:=len;
        v:=nextadj(g, child, v);
        len:=current_len;
     END
     len:=maxlen;
ENDP;{dfs}
```

算法的主要过程是对图g进行深度优先遍历，基图g以邻接表的形式存储，则时间复杂

度为$O(n+e)$。

例题7-6　（清华大学2000年试题）

请回答下列关于图（Graph）的一些问题：

① 有n个顶点的有向强连通图最多有多少条边？最少有多少条边？

② 表示一个有1000个顶点，1000条边的有向图的邻接矩阵有多少个矩阵元素？是否为稀疏矩阵？

③ 对于一个有向图，不用拓扑排序，如何判断图中是否存在环？

【分析】 有n个顶点的有向强连通图最少有n条边，如n个顶点依次首尾相接构成一个环。

关于稀疏矩阵的定义是：假若值相同的元素或者零元素在矩阵中的分布有一定的规律，则称此类矩阵为特殊矩阵；反之，称为稀疏矩阵。因此，在本题中，虽然一个有1000个顶点，1000条边的有向图的邻接矩阵中的非零元素非常少，但如果这些非零元素分布的有一定规律的话，则这一矩阵并不一定是稀疏矩阵。

除了拓扑排序算法，也可以用深度优先遍历方法判断一个有向图是否存在环。简单的适用于无向图的深度优先遍历方法并不能断定有向图中一定存在环，因为对有向图来说，深度优先遍历中遇到的回边有可能是指向深度优先森林中另一棵生成树上顶点的弧。

【解答】① 有n个顶点的有向强连通图最多有$n(n-1)$条边，最少有n条边。

② 表示一个有1000个顶点、1000条边的有向图的邻接矩阵有1000^2个矩阵元素，不一定是稀疏矩阵，因为它可能是特殊矩阵。

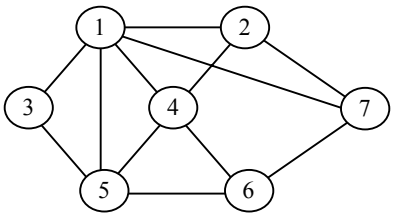
③ 对有向图进行深度优先遍历。如果从有向图上某个顶点v出发的遍历，在dfs(v)结束之前出现一条从顶点u到顶点v的回边，由于u在生成树上是v的子孙，则有向图中必定存在包含顶点v和顶点u的环。

例题7-7　（中国科学院软件研究所1999年试题）

下图中给出由7个顶点组成的无向图。从顶点1出发，对它进行深度优先遍历得到的顶点序列是＿（1）＿;而进行广度优先遍历得到的顶点序列是＿（2）＿。

（1）A. 1354267　B. 1347625　C. 1534276　D. 1247653　E. 以上答案均不正确

（2）A. 1534267　B. 1726453　C. 1354276　D. 1247653　E. 以上答案均不正确



【解答】（1）C　（2）C

例题7-8　（中国科学院软件研究所1999年试题）

从邻接矩阵 $A=\begin{bmatrix}0&1&0\\1&0&1\\0&1&0\end{bmatrix}$

可以看出，该图共有__（1）__个顶点。如果是有向图，该图共有__（2）__条弧；如果是无向图，则共有__（3）__条边。

（1）A. 9　　B. 3　　C. 6　　D. 1　　E. 以上答案均不正确

（2）A. 5　　B. 4　　C. 3　　D. 2　　E. 以上答案均不正确

（3）A. 5　　B. 4　　C.3　　D. 2　　E. 以上答案均不正确

【解答】（1）B　（2）B　（3）D

例题7-9 （中国科学院软件研究所1998年试题）

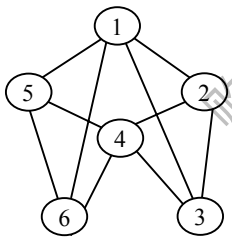
用DFS遍历一个无环有向图，并在DFS算法退栈返回时，打印出相应的面点，则输出的面点序列是_____。

A. 逆拓扑有序的　　B. 拓扑有序的　　C. 无序的

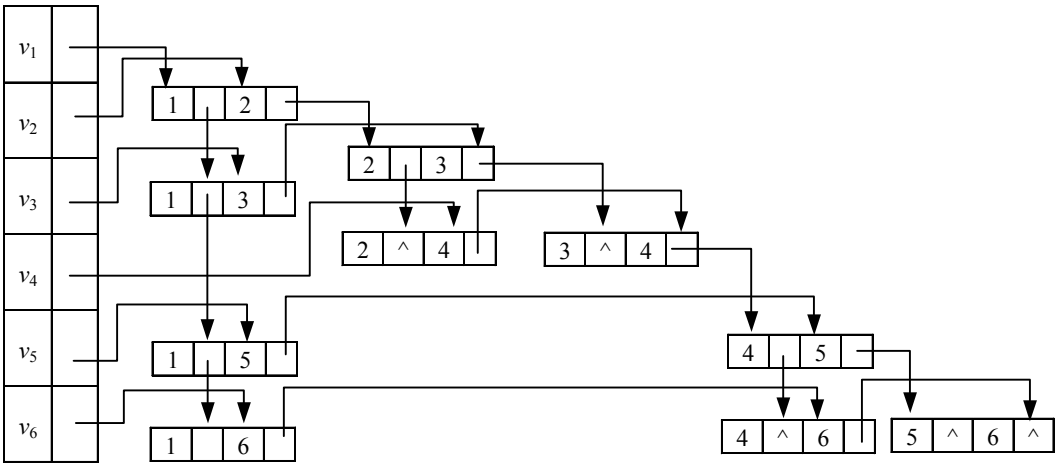
【解答】A

例题7-10 （中国科学院自动化研究所1999年试题）

画出如下图所示的无向图的邻接多重表，使得其中无向边结点中第1个顶点号小于第2个顶点号，且每个顶点的各邻接边的链接顺序为它所邻接到的顶点序号由小到大的顺序。同时请列出深度优先和广度优先遍历所得到的顶点序列和边序列。



【解答】邻接多重表如下图所示。



深度优先遍历所得到的顶点序列和边序列。

顶点序列：1，2，3，4，5，6

边序列：（1，2）（2，3）（3，4）（4，5）（5，6）

广度优先遍历所得到的顶点序列和边序列。

顶点序列：1，2，3，5，6，4

边序列：（1，2）（1，3）（1，5）（1，6）（2，4）

例题7-11 （北京邮电大学1998年试题）

已知有8个结点值为A、B、C、D、E、F、G和H的无向图，其邻接矩阵的存储结构见下表，由此结构从A结点开始深度优先遍历，得到的结点序列是_____。

	A	B	C	D	E	F	G	H
A	0	1	0	1	0	0	0	0
B	1	0	1	0	1	1	1	0
C	0	1	0	1	0	0	0	0
D	1	0	1	0	0	0	1	0
E	0	1	0	0	0	0	0	1
F	0	1	0	0	0	0	1	1
G	0	1	0	1	0	1	0	1
H	0	0	0	0	1	1	1	0

供选择的答案：

A. ABCDGHFE

B. ABCDGFHE

C. ABGHFECD

D. ABFHEGDC

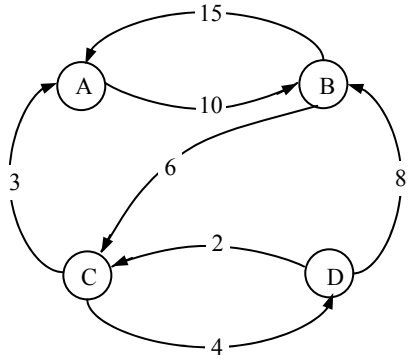
E. ABEHFGDC

F. ABEHGFCD

【解答】B

例题7-12 （北京邮电大学1998年试题）

某乡有A、B、C和D共4个村庄，如下图所示。图中边上的数值W_{ij}即为从i村庄到j村庄的距离，现在要在乡里建立中心俱乐部，其选址应使得离中心最远的村庄离俱乐部最近。



① 请写出各村庄之间的最短距离矩阵；

② 写出该中心俱乐部应设在哪个村庄，以及各村庄到中心俱乐部的路径和路径长度。

【解答】① 各村庄之间最短距离：

	A	B	C	D
A	-	10	16	20
B	9	-	6	10
C	3	12	-	4
D	5	8	2	-

② 中心俱乐部应设在A村庄，各村庄到A的路径如下：

B：B->C->A　　长度9

C：C->A　　长度3

D：D->C->A　长度5

例题7-13 （北京邮电大学1998年试题）

判断正误：

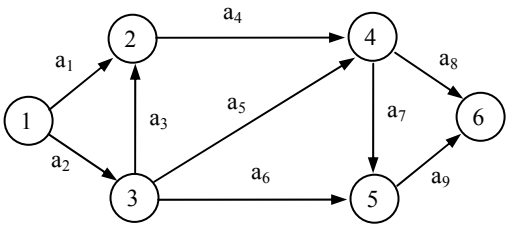
有n个顶点的无向图，采用邻接矩阵表示，图中的边数等于邻接矩阵中非零元素之和的一半。

【解答】错误

例题7-14 （北京工业大学1998年试题）

计算下图所示的AOE网中各顶点所表示的事件发生时间ve(j)、vl(j)，各边所表示活动的开始时间e(i)、l(i)，并找出其关键路径。

其中：a_1=2　a_2=3　a_3=3　a_4=5　a_5=9　a_6=4　a_7=6　a_8=2　a_9=3



【解答】

ve(1)=0
ve(2)=max{a_1，a_2+a_3}=3+3=6
ve(3)=3
ve(4)=max{6+a_4，3+a_5}=3+9=12
ve(5)=12+6=18
ve(6)=18+3=21
vl(6)=ve(6)=21
vl(5)=21-3=18
vl(4)=18-6=12
vl(3)=12-9=3
vl(2)=12-5=7
vl(1)=3-3=0
e(a_1)=ve(1)=0
e(a_2)=0
e(a_3)=ve(3)=3
e(a_4)=ve(2)=6
e(a_3)=ve(3)=3
e(a_6)=3
e(a_7)=ve(4)=12
e(a_8)=12
e(a_9)=ve(5)=18
l(a_9)=vl(6)-a_9=21-3=18
l(a_8)=21-2=19
l(a_7)=vl(5)-a_7=18-6=12
l(a_6)=vl(5)-a_6=18-4=14
l(a_5)=vl(4)-a_5=12-9=3
l(a_4)=vl(4)-a_4=12-5=7
l(a_3)=vl(2)-a_3=7-3=4
l(a_2)=vl(3)-a_2=3-3=0
l(a_1)=vl(2)-a_1=7-2=5

所以，图中关键路径为a_2，a_5，a_7，a_9，即(v_1，v_3，v_4，v_5，v_6)。

例题7-15　（中国科学院计算机网络信息中心2001年试题）

单选题：

*n*个顶点的强连通图的邻接矩阵中至少有_____个非零元素。

A．*n*-1　　B．*n*　　C．2*n*-2　　D．2*n*

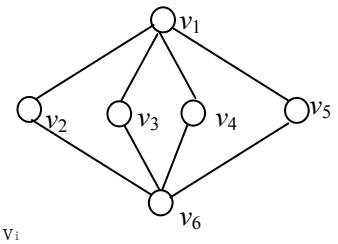
【解答】B

例题7-16　（中国科学院计算机网络信息中心2001年试题）

程序阅读题：

以下伪代码是一个广度优先搜索算法，请给出该算法在下图上执行BFS(G，1)的结果，指出其中的错误并修改代码使得输出正确。

```
void BFS(ALGraph *G, int k) {
  //G是图的邻接表，k是顶点序号
  InitQueue(&Q);//设置空队列Q
  EnQueue(&Q, k);//k入队
  while (! QueueEmpty(&Q) ) {
      i=DeQueue( &Q);//队首元素出队
      visited[i]=TRUE;//设置访问标记
      printf("%c", G->adjlist[i];vertex); //访问顶点vi
      for(p=G->adjlist[i].firstedge;p; p=p->next )
         // 依次搜索顶点vi的相邻顶点vj（设p->adjvex=j）
         if( !visited[p->adjvex])
            EnQueue(&Q, p->adjvex); // 顶点vj的序号入队
      } // end while
 } // end BFS
```



【解答】

转出序列为：$v_1v_2v_3v_4v_5v_6v_6v_6v_6$

错误：当v_2，v_3，v_4，v_5出队时均执行了v_6入队操作，导致了对v_6的重复入队和浮向。

修改：方法一，出队时判别v_i是否浮向过，已访问时不转出，但这样仍不能保证同一个顶点的多次入队，但结果仍算正确；方法二，入队时做浮向标记号，使已标记的顶点不会重复入队。

例题7-17 （中国科学院计算机网络信息中心2001年试题）

算法题：

设有向图$G=(V, E)$中的顶点表示通信结点，边表示通信链路，每条边$(u, v)\in E$均对应一个实数值$0\leqslant r(u, v)\leqslant 1$，它表示从顶点$u$到顶点$v$的通信链路不中断的概率（即通信链路的可靠性），假设这些概率是相互独立的，试写一有效算法找出指定顶点对之间的最可靠的通信路径，并给出时间复杂度分析。

【解答】可利用单源最短路径的Dijkstra算法或每一顶点对间的最短路径的Floyd算法，但应作以下修改：

① Dijkstra算法中，条件dist[j]+cost[j, i]<dist[i]应改为：

dist[j]*cost[j, i]>dist[i]

即求给定顶点s到t的最可靠路径，若为：

$S=v_{i1}, v_{i2}, \cdots v_{ik}=t$，则$r(v_{i1}, v_{i2})*r(v_{i2}, v_{i3})*\cdots*r(v_{ik-1}, v_{ik})$最大，这里$r(j, i)$相当于Dijkstra算法中的cost[j, i]，时间复杂度为$O(n^2)$。

② Floyd算法中，条件length[i, k]+length[k, j]<length[i, j]应改为

length[i, k]*length[k, j]>length[i, j]，时间复杂度为$O(n^3)$。

例题7-18 （中国科学院计算机网络信息中心2001年试题）

当各边上的权值_____时，BFS算法可用来解决单源最短路径问题。

A．均相等　　B．均互不相等　　C．不一定相等

【解答】A

例题7-19　（中国科学院计算机网络信息中心2000年试题）

试以逆邻接表为存储结构，通过每次删除出度为零的顶点及其入边来写一拓扑排序算法，要求输出的顶点序列是拓扑有序序列。

【解答】伪代码算法如下：

```
Topsort(G){
        while(G中去出度为0的顶点)do
            从G中选一出度为0的顶点U，并将其入栈S;
            从G中删去U及U的所有入边; //
}
if(入栈S的顶点数<|V|)
        Error("G有环");
else依次将S退格并转出直至栈空;
}
```

算法中应设置出度向量，及保存当前所有出度为零的顶点的栈（或队列）。

例题7-20　（华北计算技术研究所2001年试题）

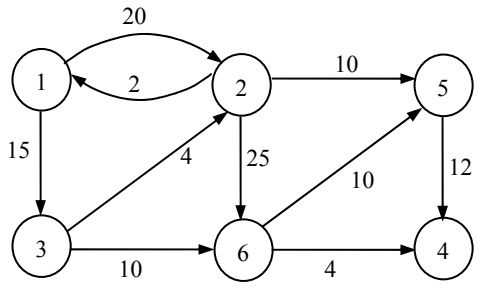
已知加权有向图G的邻接矩阵如下：

$$A=\begin{bmatrix}\infty & 20 & 15 & \infty & \infty & \infty\\ 2 & \infty & \infty & \infty & 10 & 25\\ \infty & 4 & \infty & \infty & \infty & 10\\ \infty & \infty & \infty & \infty & \infty & \infty\\ \infty & \infty & \infty & 12 & \infty & \infty\\ \infty & \infty & \infty & 4 & 10 & \infty\end{bmatrix}$$

（1）画出该有向图G;

（2）试利用Dijkstra算法求G中从顶点1到其他各顶点间的最短路径，并给出运算过程中dist向量的变化状态。

【解答】（1）有向图G如下所示：



（2）利用Dijkstra算法求G。

终点	从v_1到各终点的dist值和最短路径				
V_1	∞	∞	∞	∞	∞
V_3	15 (1，3)				
V_4	∞	∞	∞	29* (1，3，6，4)	
V_5	∞	∞	29 (1，3，2，5)	29 (1，3，2，5)	29* (1，3，2，5)
V_6		25 (1，3，6)	25 (1，3，6)		
V_2	20 (1，2)	19 (1，3，2)			
V_j	V_3	V_2	V_6	V_4	V_5

例题7-21 （北京大学2000年试题）

设G为带权有向图，$G=(K, E)$，其中$K=\{K_1, K_2, K_3, K_4, K_5, K_6, K_7\}$，用$\langle K_i, K_j, W_1\rangle$表示结点$K_i$到结点$K_j$的权为$W_1$的边。$E=\{\langle K_1, K_2, 10\rangle, \langle K_1, K_3, 8\rangle, \langle K_1, K_4, 20\rangle, \langle K_2, K_4, 5\rangle, \langle K_3, K_5, 20\rangle, \langle K_3, K_4, 7\rangle, \langle K_4, K_6, 6\rangle, \langle K_5, K_4, 3\rangle, \langle K_5, K_6, 9\rangle, \langle K_5, K_7, 2\rangle, \langle K_6, K_7, 2\rangle\}$，给出从$K_1$到$K_7$的所有最短路径及其长度。

【解答】

$K1$到$K7$的最短路径有：$K_1\rightarrow K_3\rightarrow K_4\rightarrow K_6\rightarrow K_7$　　>3（长度）

$K_1\rightarrow K_3\rightarrow K_4\rightarrow K_6\rightarrow K_7$　　>3（长度）

例题7-22 （中国人民大学2002年试题）

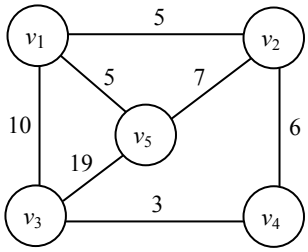
以下算法是统计无向图的连通分量个数的算法，请在留空处填入适当内容：

```
void gnum(GRAPH g, int*num)
  {int i, visited[N]; /*N是代表图中最大顶点数的符号常量*/
    for(i=0;i<g.vexnum;i++)visited[i]=0;
        if(    ①    )
          {dfs(g, i);      /*从顶点i出发，深度遍历图。*/
            ②    ;
          }
}
```

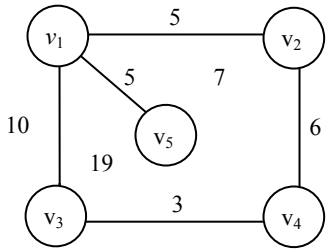
【解答】① !visited[i]　② (*num)++

例题7-23 （中国人民大学2002年试题）

对下图所示的带权图，用普里姆算法画出该图的最小生成树的生成过程。



【解答】



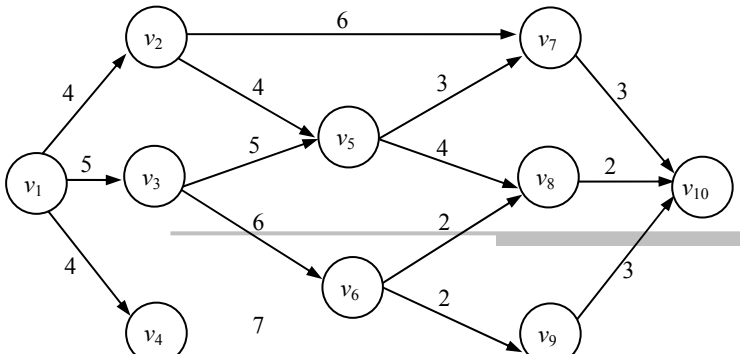
粗笔所示路径为关键路径。

Prim生成过程：

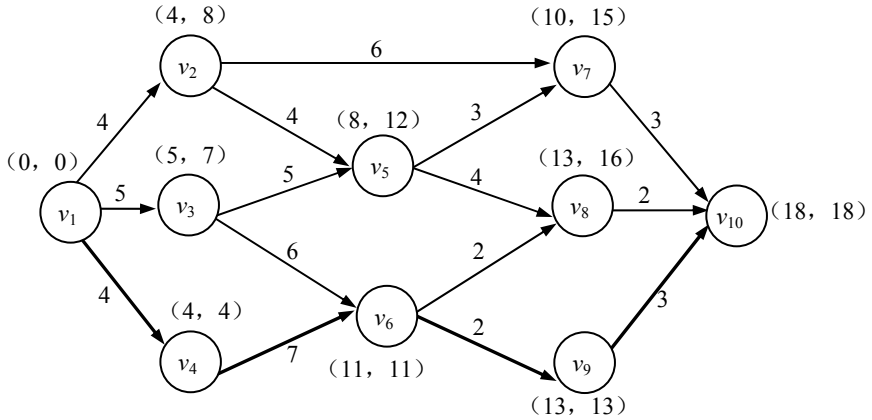
V / closedge	v_1	v_2	v_4	v_5	U	$V-U$
vex	v_3		v_3	v_3	$\{v_3\}$	$\{v_1, v_2, v_4, v_5\}$
lowcost	10		3	19		
vex	v_3		v_2	v_3	$\{v_3, v_4\}$	$\{v_1, v_2, v_5\}$
lowcost	10		6	19		
vex	v_3		v_2	v_2	$\{v_3, v_4, v_2\}$	$\{v_1, v_5\}$
lowcost	10		6	7		
vex	v_5	v_5			$\{v_3, v_4, v_2, v_5\}$	$\{v_1\}$
lowcost	5	7				
vex	0	0	0	0	$\{v_1, v_2, v_3, v_4, v_5\}$	φ
lowcost						

例题7-24 （中国人民大学2002年试题）

画出下图所示的AOE网的所有关键路径，并在各顶点的上、下方标出各顶点事件的最早和最迟发生时间。



【解答】



粗笔所示路径即为关键路径。

例题7-25 （北京科技大学2002年试题）

构造无向连通网的最小生成树通常有哪两个典型的算法？

【解答】普里姆（Prim）算法和克鲁斯卡尔（Kruskal）算法

例题7-26 （北京科技大学2002年试题）

设一个无向网G的邻接矩阵A如下：

$$A=\begin{bmatrix} \infty & \infty & 4 & 2 & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & 1 & 7 & \infty & \infty \\ 4 & \infty & \infty & \infty & \infty & \infty & 6 & \infty \\ 2 & \infty & \infty & \infty & 3 & \infty & 5 & \infty \\ \infty & 1 & \infty & 3 & \infty & \infty & \infty & 8 \\ \infty & 7 & \infty & \infty & \infty & \infty & \infty & 9 \\ \infty & \infty & 6 & 5 & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & 8 & 9 & \infty & \infty \end{bmatrix}$$

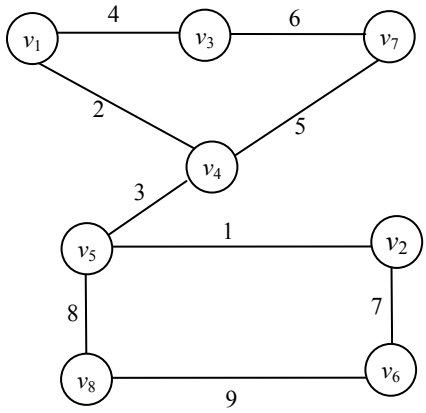
（1）请根据给定的邻接矩阵A画出网G的逻辑结构（G中顶点用v_1~v_8表示）；

（2）写出从顶点v_1出发、按“深度优先”和“广度优先”搜索方法遍历网G所得到的顶点序列；

（3）从顶点v_1出发，按照求最小生成树的Prim算法，画出网G的一棵最小生成树。

【解答】

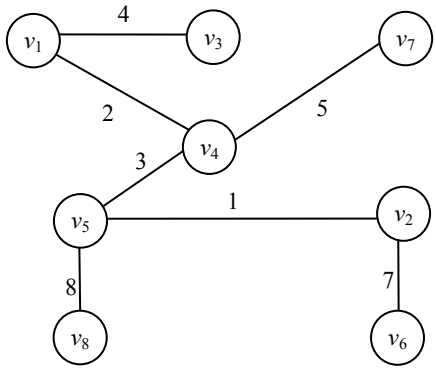
（1）



（2）深度优先：v_1 v_3 v_4 v_7 v_5 v_2 v_6 v_8

广度优先：v_1 v_3 v_4 v_7 v_5 v_2 v_8 v_6

（3）



例题7-27 （武汉理工大学2002年试题）

选择题：

一个加权连通无向图的最小生成树可以使用（1）生成；一项工程完工所需的最少时间等于某个（2）。

供选择的答案：

（1）A. Hash算法　　B. Dijkstra算法　　C. Prim算法　　D. Huffman算法

（2）A. AOE网中源点到汇点事件最多的路径的长度

B. AOE网中源点到汇点的最长路径的长度

C. AOE网中源点到汇点的最短路径的长度

D. AOE网中源点到汇点活动最多的路径的长度

【解答】（1）C　　（2）B

例题7-28 （武汉理工大学2002年试题）

判断题：

邻接多重表表示法对于有向图和无向图的存储都适用。

【解答】正确。

例题7-29 （武汉理工大学2002年试题）

判断题：

每个加权连通无向图的最小生成树都是惟一的。

【解答】错误。

例题7-30 （武汉理工大学2002年试题）

判断题：

不是所有的AOV网都有一个拓扑序列。

【解答】正确。

例题7-31 （武汉理工大学2002年试题）

简答题：

试述一种判定有向图G中是否有圈（回路）的方法。

【解答】利用拓扑序列算法可以判定G是否有圈，在拓扑排序中若输出之后余下的顶点均有前驱，则说明得到部分顶点的拓扑有序序列，网中存在着有向回路。

例题7-32 （南京理工大学2002年试题）

选择题：

无向图$G=(V, A)$，其中$V=\{a, b, c, d, e\}$，

$A=\{\langle a, b\rangle, \langle a, c\rangle, \langle d, c\rangle, \langle d, e\rangle, \langle b, e\rangle, \langle c, e\rangle\}$

对该图进行拓扑排序，下面序列中哪一个不是拓扑序列。

A. a，d，c，b，e　　B. d，a，b，c，e

C. a，b，d，c，e　　D. a，b，c，d，e

【解答】D

例题7-33 （南京理工大学2002年试题）

对给定的有6个顶点的无向图的邻接矩阵如下：

（1）画出从v_1出发的深度优先生成树和广度优先生成树；

（2）填写用普里姆算法构造最小生成树辅助数组中各分量的值；

（3）画出最小生成树；

（4）画出邻接表存储结构；

（5）写出Prim算法和Kruskal算法的时间复杂度。

$$\begin{bmatrix} \infty & 6 & 1 & 5 & \infty & \infty \\ 6 & \infty & 5 & \infty & 3 & \infty \\ 1 & 5 & \infty & 5 & 6 & 4 \\ 5 & \infty & 5 & \infty & \infty & 2 \\ \infty & 3 & 6 & \infty & \infty & 6 \\ \infty & \infty & 4 & 6 & 6 & \infty \end{bmatrix}$$

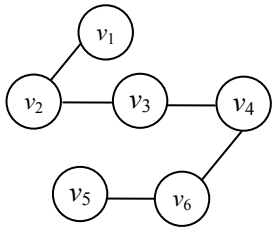
closedge	v_2	v_3	v_4	v_5	v_6
Adjvex Lowcost					
Adjvex Lowcost					
Adjvex Lowcost					
Adjvex Lowcost					

（续表）

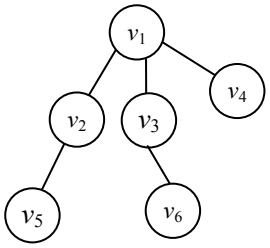
closedge	v_2	v_3	v_4	v_5	v_6
Adjvex Lowcost					
Adjvex Lowcost					

【解答】

（1）深度优先



广度优先



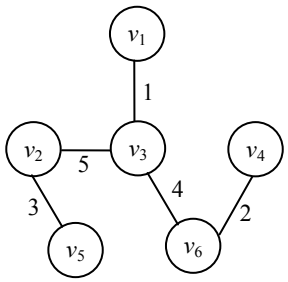
（2）

Closedge	v_2	v_3	v_4	v_5	v_6
Adjvex Lowcost	v_1 6	v_1 1	v_1 5		
Adjvex Lowcost	v_3 5	0	v_1 4	v_3 6	v_3 4
Adjvex Lowcost	v_3 5	0	v_6 2	v_3 6	0

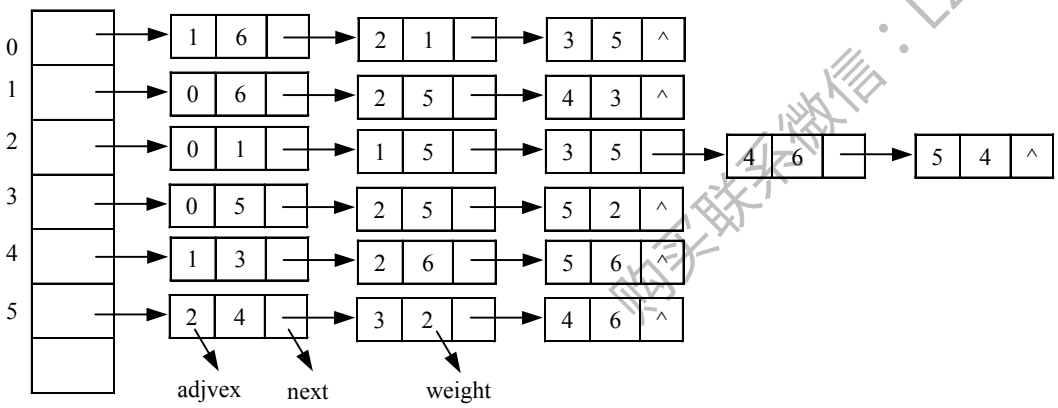
Adjvex Lowcost	v_3 5	0	0	v_3 6	0
Adjvex Lowcost	0	0	0	v_2 3	0
Adjvex Lowcost	0	0	0	0	0

（3）

错误！



（4）



（5）Prim算法的时间复杂度为$O(n^2)$，n为顶点数

Kruskal算法的时间复杂度为$O(n\text{Log}n)$。

例题7-34 （武汉大学2002年试题）

名词解释：

图

【解答】图是图形结构的简称，它是一种复杂的非线性数据结构。图由两个集合V和E组合，记为$G=(V, E)$，其中V是顶点的有穷非空集合，E是V中顶点偶对的有穷集合，这些顶点偶对称为边。

7.5　自 测 题

自测题7-1

对于一个使用邻接表存储的带权有向图*G*，试利用深度优先搜索方法，对该图中所有顶点进行拓扑排序。若邻接表的数据类型定义为Graph，则算法的首部为：

```
FUNCTION dfs_toposort(G:Graph):boolean;
```

若函数返回true，表示拓扑结构排序成功，图中不存在环；若函数返回false，则图中存在环，拓扑排序不成功。在这个算法中嵌套调用一个递归的深度优先搜索算法为：

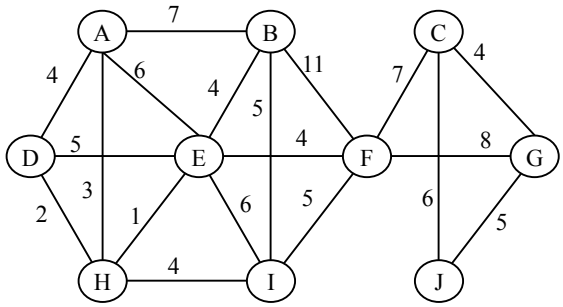
```
PROCEDURE dfs(G:Graph;V:vtxnum);
```

在遍历图的同时进行拓扑排序。其中，vtxnum是顶点号。

① 给出该图的邻接表定义；
② 定义在算法中使用的全局辅助数组；
③ 写出拓扑排序的算法。

自测题7-2

用深度优先搜索遍历如下图所示的无向图，试给出以A为起点的顶点访问序列（同一个顶点的多个邻点，按字母顺序访问），并给出一棵最小生成树。



自测题7-3

*G*是一个非连通无向图，共有28条边，则该图至少有_____个顶点。

A. 6　　B. 7　　C. 8　　D. 9　　E. 10　　F. 11

自测题7-4

在下列两种求图的最小生成树的算法中，_____算法适合于求边稀疏的网的最小生成树。

A. PRIM　　B. KRUSKAL

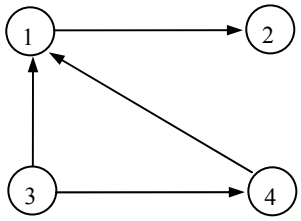
自测题7-5

在一个有n个顶点的无向网中，有$O(n^{1.5}*\log_2 n)$，则应该选用_____算法来求这个网的最小生成树，从而使计算时间较少。

A. PRIM　　B. KRUSKAL

自测题7-6

请给出如下图所示的邻接矩阵、邻接表、逆邻接表和十字链表。



自测题7-7

判断正误：

在n个结点的无向图中，若边数>n-1，则该图必是连通图。

自测题7-8

判断正误：

若一个有向图的邻接矩阵中对角线以下元素均为零，则该图的拓扑有序序列必定存在。

自测题7-9

图的遍历方式有_____和_____两种。

自测题7-10

已知AOE网中顶点v_1、v_2、v_3、v_4、v_5、v_6和v_7分别表示7个事件，有向线段a_1、a_2、a_3、a_4、a_5、a_6、a_7、a_8、a_9和a_{10}分别表示10个活动，线段旁的数值表示每个活动花费的天数，如下图所示。请分别计算出各事件的最早发生时间、各事件的最晚发生时间、各活动的最早开始时间、各活动的最晚开始时间、各活动的松驰时间，分别填入表7.1和表7.2中。用顶点序列表示出关键路径，给出关键活动。

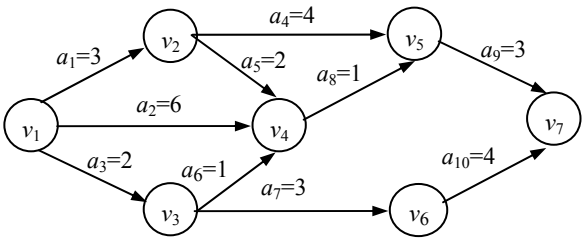


表7.1

事件	V_1	V_2	V_3	V_4	V_5	V_6	V_7
最早发生时间							
最晚发生时间							

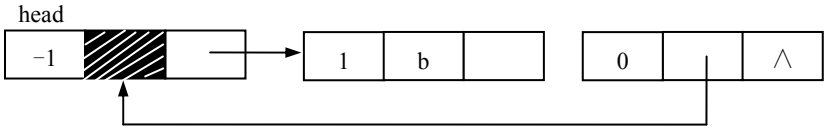
表7.2

活动	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}
最早发生时间										
最晚开始时间										
松驰时间										

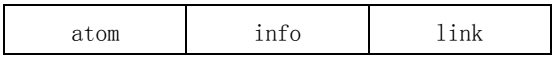
自测题7-11

下图所示的结构是一个(　　)。

A. 线性表　　B. 树形结构　　C. 图结构　　D. 广义表



图中结点结构为：



自测题7-12

有一带权无向图的顶点集合为{V_1，V_2，V_3，V_4，V_5，V_6，V_7}。已知其邻接矩阵的三元组表示形式见下表。

8	8	20
1	2	12
1	5	2
2	1	12
2	6	3
2	8	5
3	4	8
3	5	2
3	6	4
4	3	8
4	5	10
4	7	8
5	1	2

5	3	2
5	4	10
6	2	3
6	3	4
6	7	7
7	4	8
7	6	7
8	2	5

（1）请画出该无向图的邻接表。

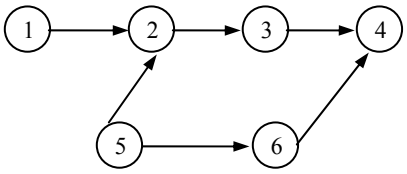
（2）请画出所有可能的最小花费生成树。

（3）根据我给的邻接表分别写出从v_1出发进行深度优先遍历与广度优先遍历的顶点序列。

（4）v_1与v_2的最短路径是什么？

自测题7-13

试列出下图中全部可能的拓扑排序序列。

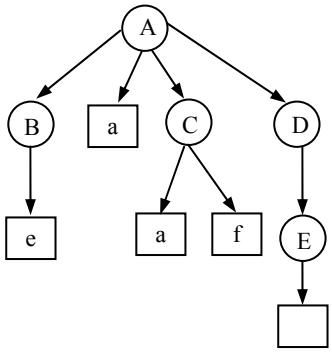


自测题7-14

假设以邻接矩阵作图的存储结构，编写算法判别在给定的有向图中是否存在一个简单有向回路。若存在，则以顶点序列的方式输出该回路（找到一条即可）。（注：图中不存在顶点到自己的弧。）

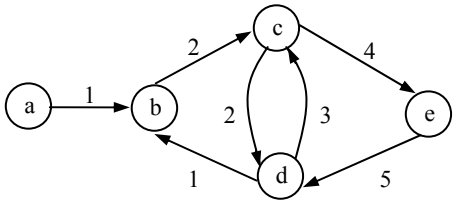
自测题7-15

将下图所示的有根有序的有向图转换成一个（或一组）广义表。



自测题7-16

求解下图所示的有向图的有关问题。



（1）判断此有向图是否有强连通分量？若有请画出。

（2）画出此有向图的十字链表存储结构。

其顶点表结点结构为：

data	firstin	firstout

data:顶点的有关信息；

firstin:指向以该顶点为弧头的第一条边的指针；

firstout:指向以该顶点为弧尾的第一条边的指针。

其边表结点的结构为：

tailvex	headvex	weight	hlink	tlink

tailvex，headvex：分别为弧尾和弧头在图中的序号；

weight：弧上的权值；

hlink，tlink：分别为指向弧头相同和弧尾相同的下一条边的指针。

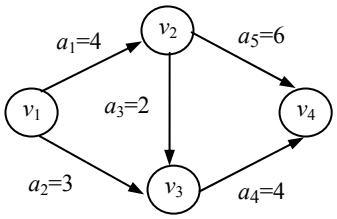
（3）设其顶点a、b、c、d、e表示一个乡的5个村庄，弧上的数值表示为两村之间的距离。

a. 求每个村庄至其他村庄的最短距离；

b. 乡内要建立一所医院问医院设在哪个村庄才能使各村离医院的距离较近。

自测题7-17

对于下图所示的事件结点网络，求出各活动可能的最早开始时间和允许的最晚完成时间，并问哪些活动是关键活动。



7.6 自测题答案

【自测题7-1】

检查一个有向图是否无环要比无向图复杂。对于无向图来说，若深度优先遍历过程中遇到回边，则必定存在环；而对于有向图来说，这条回边有可能是指向深度优先森林中另一棵生成树上顶点的弧。但是，如果从有向图上某个顶点*v*出发的遍历，在dfs(*v*)结束之前出现一条从顶点*u*到顶点*v*的回边，由于*u*在生成树上是*v*的子孙，则有向图中必定存在包含顶点*v*和顶点*u*的环。

答案：

① TYPE vexptr=1…n;
Arcptr=^arcnode;
Arcnode=RECORD
Adjvex:vtxptr;
Nextarc:arcptr;
Info:… {和弧有关的其他信息}
END;
vexnode=RECORD
Vexdata:…; {和顶点有关的其他信息}
firstarc:arcptr;
END;
Graph=ARRAY[vexptr]OF vexnode;

② 在算法中使用的全局辅助数组为：

```
visited:ARRAY vexptr OF Boolean;
{若visited[i]=true，则表示顶点i已被访问过；否则表示顶点i未被访问过}
finished:ARRAY vexptr OF Boolean;
{若finished[i]=true，则表示对顶点i的dfs搜索已经结束；否则表示对顶点i的dfs搜索未结束}
```

③ 函数dfs_toposort和过程dfs的定义如下：

```
FUNCTION dfs_toposort(G:Graph):Boolean;
flag:=true;
FOR i:=1 TO n DO visited[i]:=false;
FOR i:=1 TO n DO finished[i]:=true;
i:=1;
WHILE(flag)AND (i<=n) DO BEGIN
  IF (visited[i]=false) dfs(G, i)
       finished[i]:=true;
END
return flag;
ENDF;{dfs_toposort}
```

拓扑排序的算法如下：

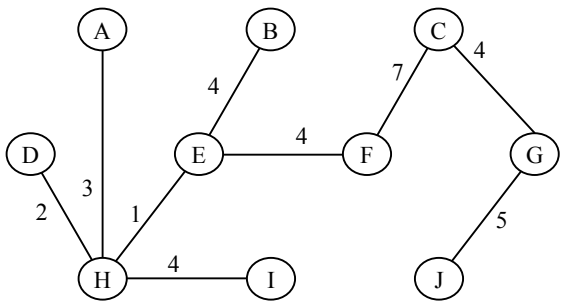
```
PROCEDURE dfs(G:Graph;V:vtxnum);
{在深度优先遍历时打印出顶点号，若图中不存在环，则打印出的全部顶点号即为有向图的顶点的拓
```

```
扑排序，否则将标志flag置为false}
write(v);
finished[v]:=false;
visited[v]:=true;
p:=Graph[V].firstarc;
WHILE(p<>NIL)DO
BEGIN
     IF (visited[P^.adjvex]=true) AND (finished[p^.adjvex]=false)
     flag:=false;
     ELSE IF (visited[p^.adjvex]=false)
          BEGIN
          Dfs(G, P^.adjvex);
          finished[p^.adjvex]:=true;
          END
     p:=p^.nextarc;
     END
ENDP;{dfs}
```

【自测题7-2】

顶点访问序列：A B E D H I F C G J

一棵最小生成树如下图所示。



【自测题7-3】

答案为D。

【自测题7-4】

答案为B。

【自测题7-5】

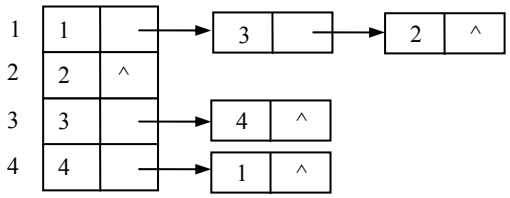
答案为B。

【自测题7-6】

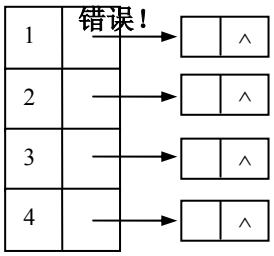
邻接矩阵：$\begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}$

邻接表如下图所示。

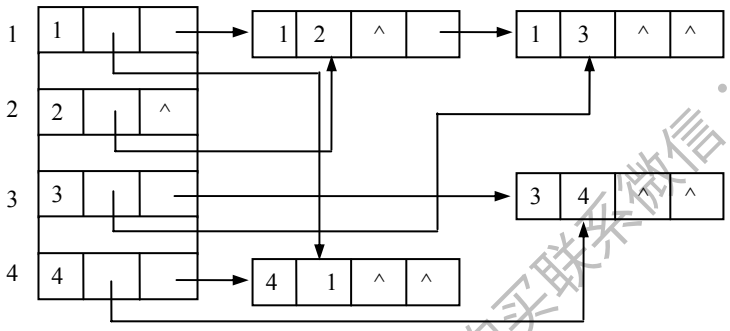
错误！



逆邻接表如下图所示。



十字链表如下图所示。



【自测题7-7】

该图可能包含多个连通子图，但其本身不是连通图。

答案为错误。

【自测题7-8】

答案为正确。

【自测题7-9】

答案为深度优先，广度优先。

【自测题7-10】

各时间填入后见表7.3和表7.4。

表7.3

事件	V_1	V_2	V_3	V_4	V_5	V_6	V_7
最早发生时间	0	3	2	6	7	5	10
最晚发生时间	0	3	3	6	7	6	10

表7.4

活动	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}
最早发生时间	0	0	0	3	3	2	2	6	7	5
最晚开始时间	0	0	1	3	4	5	3	6	7	6
松 驰 时 间	0	0	1	0	1	3	1	0	0	1

关键路径：V_1 V_2 V_5 V_7

V_1 V_4 V_5 V_7

关键活动：a_1 a_2 a_4 a_8 a_9

【自测题7-11】

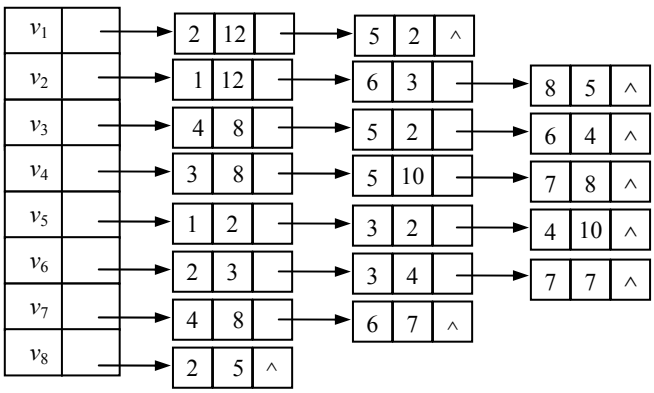
答案为D。

【自测题7-12】

每个三元组（顶点，顶点，权重）代表一条边，由此可以得到邻接矩阵和邻接表。

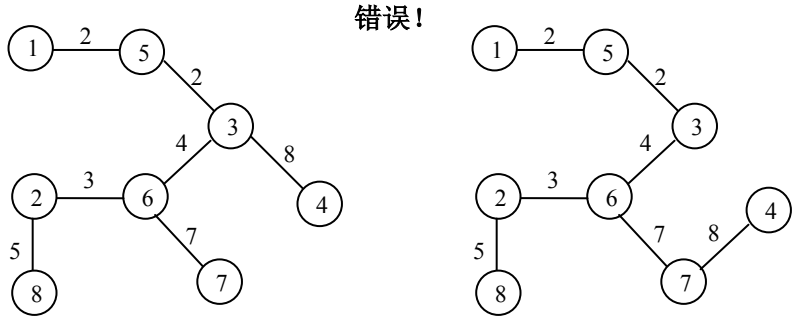
答案：

（1）邻接表如下图所示。



（2）最小花费生成树如下图所示。

错误！



（3）深度优先遍历顶点序列：1，2，6，3，4，5，7，8
广度优先遍历顶点序列：1，2，5，6，8，3，4，7
（4）v_1到v_2的最短路径是：

$v_1 -> v_5 -> v_3 -> v_6 -> v_2$

长度为11。

【自测题7-13】
按以下算法可以得到所有拓扑排序序列。
（1）在有向图中选一个没有前驱的顶点，并输出。
（2）在图中删除该顶点和相关的边。
重复（1）（2）。
答案：

1 5 2 3 6 4
1 5 2 6 3 4
1 5 6 2 3 4
5 6 1 2 3 4
5 1 6 2 3 4
5 1 2 6 3 4
5 1 2 3 6 4

【自测题7-14】
采用深度优先遍历，若发现回路则输出。
答案：

```
CONST MAX=…;
VAR matrix: ARRAY[1..MAX, 1..MAX]OF INTEGER;
    visited:ARRAY[1..MAX]OF BOOLEAN;
vex:    ARRAY[1..MAX+1] OF INTEGER;
p,i:      INTEGER;
found:BOOLEAN;

PROCEDURE DFS(v:INTEGER);
BEGIN
   IF found THEN return;
p:=p+1;
Vex[p]:=v;
IF visited[v] THEN BEGIN
      found:=TRUE;
      display();
      END
ELSE BEGIN
      visited[v]:=TRUE;
      FOR i:= TO max DO
           if matrix[v, i]>0 THEN DFS(i);
END;
p:=p-1;
```

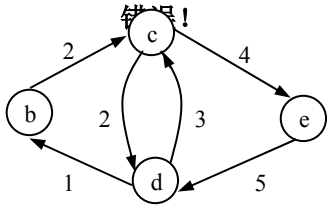
```
   visited[v]:=FALSE;
END;

PROCEDURE display;
VAR    v, i:=INTEGER;
     flag:=BOOLEAN;
BEGIN
    flag:=FALSE;
     v:=vex[p];
     FOR i:=1 TO p DO BEGIN
          IF vex[i]=v THEN flag:=TRUE;
     IF flag THEN write(vex[i]);
END;
END;

PROCEDURE FindCircle;
VAR v:INTEGER;
BEGIN
     FOR v:=1 TO max DO visited[i]:=FALSE;
     p:=0;
     found:=FALSE;
     FOR v:=1 TO max DO DFS(v);
END;
```

【自测题7-15】

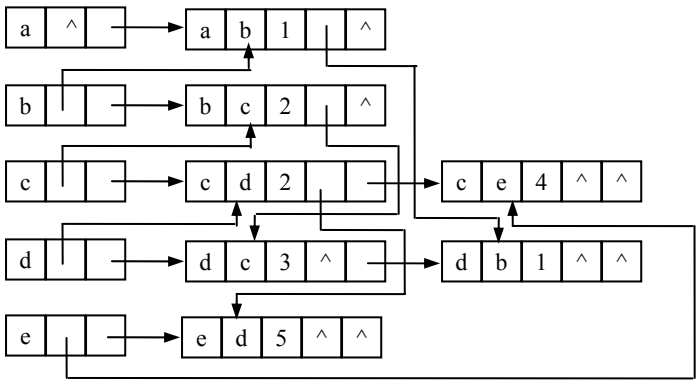
转化成的广义表为：((*e*)，*a*，(*a*，*f*)，(()))。

【自测题7-16】

（1）有强连通分量，如下图所示。



（2）有向图的十字链表存储结构如下图所示。



（3）各村之间的最近距离。

	a	b	c	d	e
a	–	1	3	5	7
b	–	–	2	4	6
c	–	3	–	2	4
d	–	1	3	–	7
e	–	6	8	5	–

医院设在b村庄比较好，因为这样各村庄到医院距离和最小。

【自测题7-17】

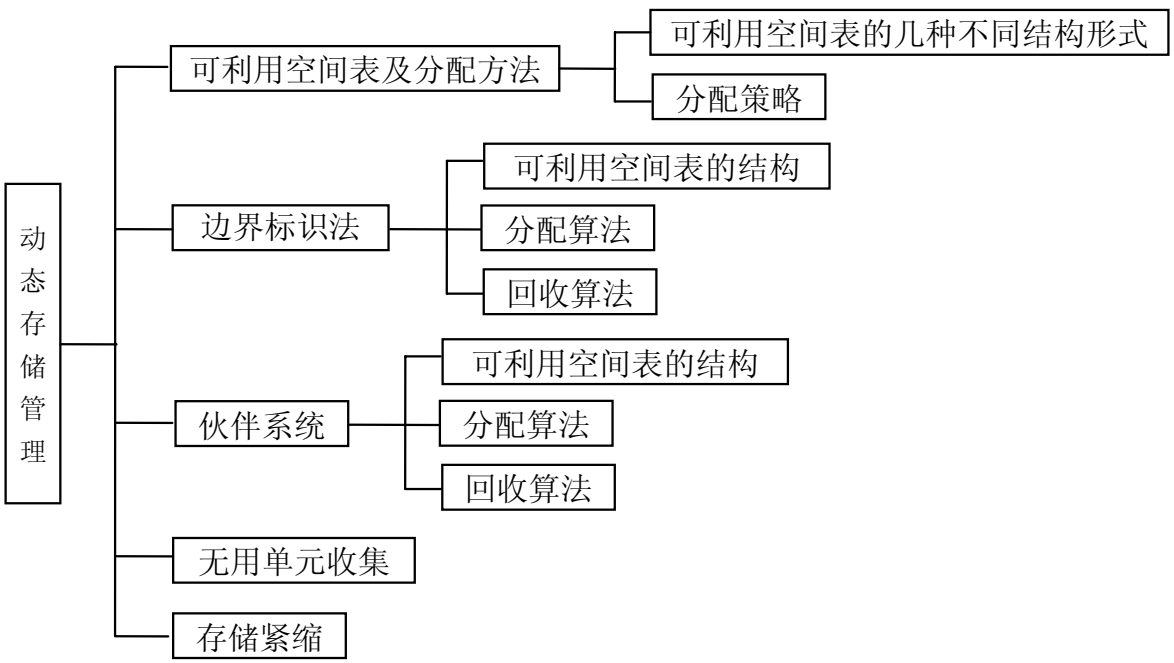
各活动的最早开始时间和允许的最晚完成时间见下表。

	a_1	a_2	a_3	a_4	a_5
最早开始时间	0	0	4	6	4
最晚完成时间	4	6	6	10	10

关键活动：a_1 a_3 a_4 a_5

第 8 章　动态存储管理

8.1　基本知识结构图



8.2　知 识 点

1. 利用可利用空间表进行动态存储分配

可利用空间表可以有3种不同的结构形式：

- 系统运行期间所有用户请求分配的存储量大小相同。
- 系统运行期间用户请求分配的存储量有若干种大小的规格。
- 系统在运行期间分配给用户的内存块的大小不固定，可以随请求而变。

第3种情况下，若当前可利用空间表中有若干可供用户使用的空闲块时，该分配哪一块？可以有3种分配策略：首次拟合法、最佳拟合法和最差拟合法。

2. 边界标识法

在每个内存区的头部和底部两个边界上分别设有标识，以标识该区域为占用块或空闲

块，使得在回收用户释放的空闲块时易于判别在物理位置上与其相邻的内存区域是否为空闲块，以便将所有地址连续的空闲存储区组合成一个尽可能大的空闲块。

3. 伙伴系统

和边界标识法类似，所不同的是：在伙伴系统中，无论是占用块或空闲块，其大小均为2的k次幂（k为某个正整数）。

4. 无用单元收集

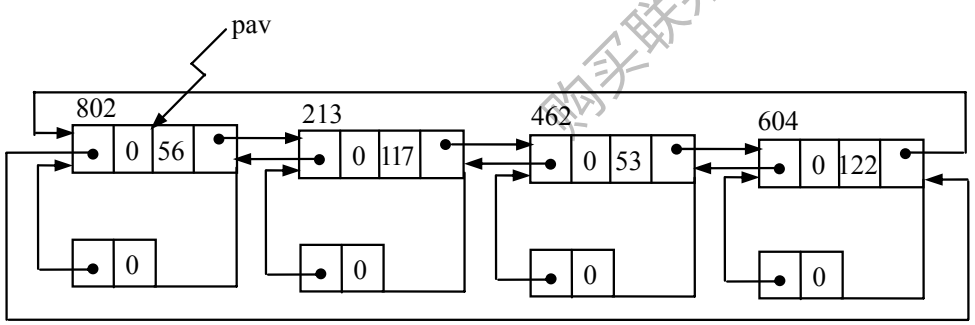
“无用单元”是指那些用户不再使用而系统没有回收的结构和变量。收集无用单元分两步进行：第一步是对所有占用结点加上标志，第二步是对整个可利用存储空间顺序扫描一遍，将所有标志域为“0”的结点链接成一个新的可利用空间表。通常有3种设置标志的算法：递归算法、非递归算法和利用表结点本身的指针域标记遍历路径的算法。

5. 存储紧缩

“紧缩存储”是在整个动态存储管理过程中，不管哪个时刻，可利用空间都是一个地址连续的存储区，在编译程序中称之为“堆”，每次分配都是这个可利用空间中划出一块。

8.3　习题及参考答案

1. 假设利用边界标识法首次适配策略分配，已知在某个时刻的可利用空间表的状态如下图所示。



（1）画出当系统回收一个起始地址为559、大小为45的空闲块之后的链表状态；

（2）画出系统继而在接受存储块大小为100的请求之后，又回收一块起始地址为515、大小为44的空闲块之后的链表状态。

注：存储块头部中大小域的值和申请分配的存储量均包括头和尾的存储空间。

【解答】注意块头中“size”域的含义和申请大小的含义。

（1）只要将原图中的604和122分别改为559和167。

（2）三块合并为一大块。将原图中第二块的始址和大小分别改为330和17；再将后两块合为一块，始址和大小分别为462和264。

2．组织成循环链表的可利用空间表可附加什么条件时，首次适配策略就转变为最佳适配策略？

【解答】空闲块按由小到大的顺序有序，且头指针恒指最小的空闲块。

3．设两个大小分别为100和200的空闲块依次顺序链接成可利用空间表。设分配一块时，该块的剩余部分在可利用空间表中保持原链接状态，试分别给出满足下列条件的申请序列：

（1）最佳适配策略能够满足全部申请而首次适配策略不能；

（2）首次适配策略能够满足全部申请而最佳适配策略不能。

【解答】可以举出多种实例序列。如下为其中一例。

（1）110，80，100

（2）110，80，90，20

4．在变长块的动态存储管理方法中，边界标识法的算法效率为什么比可利用空间表的算法效率高？

【解答】边界标识法由于在每个结点头部和底部设立了标识域，使得在回收用户释放的内存块时，很容易判别与它毗邻的内存区是否是空闲块，且不需要查询整个可利用空间表便能找到毗邻的空闲块与其合并之；再者，由于可利用空间表上结点既不需依结点大小有序，也不需依结点地址有序，所以释放块插入时也不需查找链表。因此，算法效率高。

5．考虑边界标识法的两种策略（最佳适配和首次适配）：

（1）数据结构的主要区别是什么？

（2）分配算法的主要区别是什么？

（3）回收算法的主要区别是什么？

【解答】（1）最佳适配策略下空闲块要按由小到大的顺序链接，可以不作成循环表。空闲块表头指针恒指最小空闲块。首次分配则力求使各种大小的块在循环表中均匀分布，所以经常移动头指针；

（2）无本质区别；

（3）最佳适配策略下（合并后）插入链表时必须保持表的有序性。

6．二进制地址为011011110000，大小为$(4)_{10}$的块的伙伴的二进制地址是什么？若块大小为$(16)_{10}$时又如何？

【解答】011011110100；011011100000。

7．已知一个大小为512字的内存，假设先后有6个用户提出大小分别为23，45，52，100，11和19的分配请求，此后大小为45，52和11的占用块顺序被释放。假设以伙伴系统实现动态存储管理，

（1）画出可利用空间表的初始状态；

（2）画出6个用户进入之后的链表状态以及每个用户所得存储块的起始地址；

（3）画出在回收三个用户释放的存储块之后的链表状态。

【解答】下面指出各种情形下空闲表中的块：

（1）只有一个大小为2^9的块；

（2）有大小为2^4，2^5和2^7的块各一块；

申请量	分配量	占用块始址
23	2^5	0
45	2^6	64
52	2^6	128
100	2^7	256
11	2^4	32
19	2^5	192

（3）有大小为2^5和2^6的块各两块，2^7的块1块。

8．试求一个满足以下条件的空间申请序列a_1，a_2，…，a_n：从可用空间为2^5的伙伴管理系统的初始状态开始，a_1，a_2，…，a_{n-1}均能满足，而a_n不能满足，并使$\sum_{i=1}^{n} a_i$最小。

【解答】（$2^{5-1}+1$，1）或（1，$2^{5-1}+1$）

9. 设有五个广义表：L=（L_1，L_3），L_1=（L_2，L_3，L_4），L_2=（L_3），L_3=（ ），L_4=（L_2）。若利用访问计数器实现存储管理，则需对每个表或子表添加一个表头结点，并在其中设一计数域。

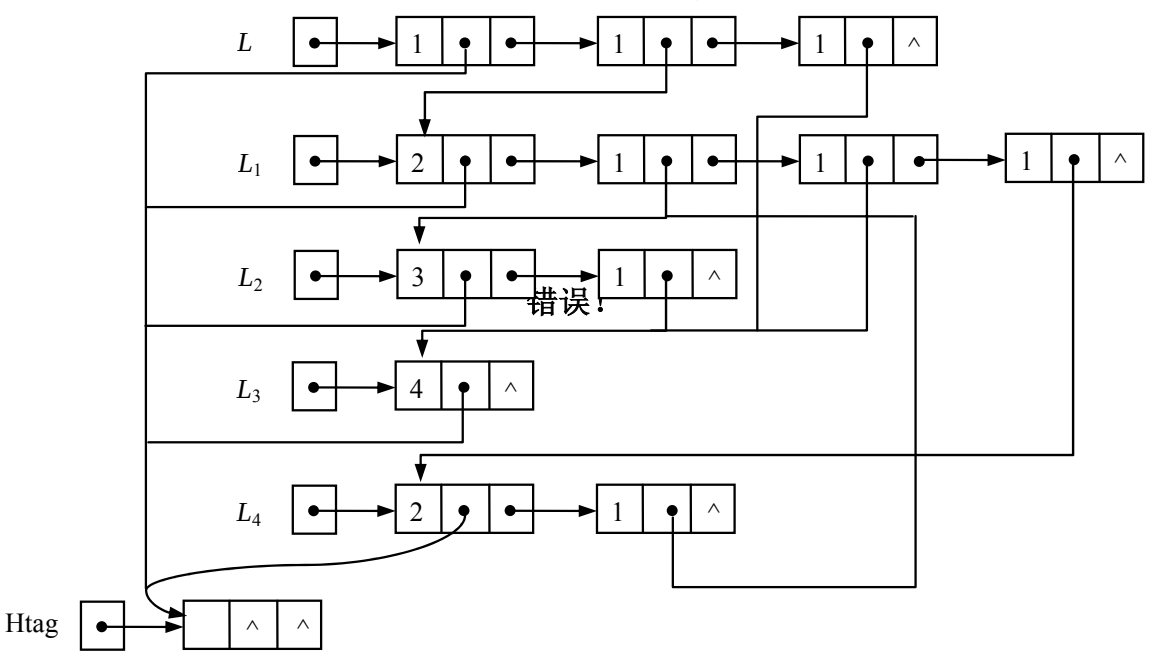
（1）试画出表L的带计数器的存储结构；

（2）从表L中删除子表L_1时，链表中哪些结点可以释放？各子表的计数域怎样变化？

（3）若L_2=（L_3，L_4），将会出现什么现象？

【解答】数据结构的设计必须综合考虑操作的实现。

（1）如下图所示。



（2）表L中首元结点可以释放，即将它从链表中删除。将表L_1的头结点计数域减1。

（3）形成间接递归。若不在实现时进行特殊处理，当删除L_2时会出现空间不能回收的不一致现象。

10．考虑空间释放遵从“最后分配者最先释放”规则的动态存储管理问题，并设每个空间申请中都指定所申请的空闲块大小。

（1）设计一个适当的数据结构实现动态存储管理；

（2）写一个为大小为n的空间申请分配存储块的算法。

【分析】利用栈先进后出的特性，将分配后的空闲块仍按原序排列，释放后可满足“最后分配者最先释放”原则。

【解答】算法如下：

```
typedef struct {
                    char *start;
                    int size;
                } fmblock;   //空闲块类型
char *Malloc_Fdlf(int n)//遵循最后分配者最先释放规则的内存分配算法
{
    while(Gettop(S，b)&&b.size<n)
    {
        Pop(S，b);
        Push(T，b);//从栈顶逐个取出空闲块进行比较
    }
    if(StackEmpty(S)) return NULL;//没有大小足够的空闲块
    Pop(S，b);
    b.size−=n;
    if(b.size) Push(S，{b.start+n，b.size});//分割空闲块
    while(!StackEmpty(T))
    {
        Pop(T，a);
        Push(S，a);
    }/恢复原来次序
return b.start;
}//Malloc_Fdlf
mem_init()//初始化过程
{
...
    InitStack(S);InitStack(T);   //S和T的元素都是fmblock类型
    Push(S，{MemStart，MemLen});   //一开始，栈中只有一个内存整块
...
}//main
```

11．同10题条件，写一个回收释放块的算法。

【解答】算法如下：

```
void Free_Fdlf(char *addr，int n)//与上一题对应的释放算法
{
    while(Gettop(S，b)&&b.start<addr)
```

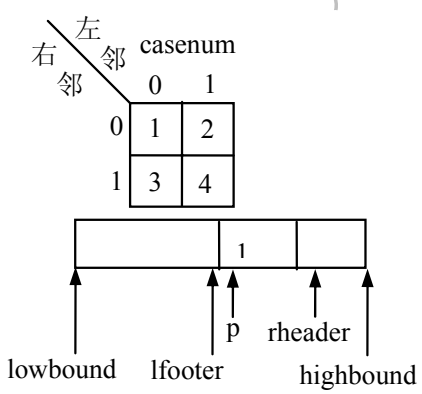
```
    {
        Pop(S，b);
        Push(T，b);
    }//在按地址排序的栈中找到合适的插入位置
    if(Gettop(T，b)&&(b.start+b.size==addr)) //可以与上邻块合并
    {
        Pop(T，b);
        addr=b.start;n+=b.size;
    }
    if(Gettop(S，b)&&(addr+n==b.start)) //可以与下邻块合并
    {
        Pop(S，b);
        n+=b.size;
    }
    Push(S，{addr，n});//插入到空闲块栈中
    while(!StackEmpty(T))
    {
        Pop(T，b);
        Push(S，b);
    }//恢复原来次序
}//Free_Fdlf
```

12．试完成边界标识法和依首次适配策略进行分配相应的回收释放块的算法。

【分析】被管理的存储空间无论大小，都是有界的。忘记处理边界情况是易犯的错误。利用下面的函数，可以使回收算法简洁清晰，关键在于利用适当的数据结构以简化情况判断。

【解答】算法如下：

```
cases dealloctype (blockptr p) {
    // cases 取值1..4，分别表示4种情形。
    对casenum [0..1，  0..1]赋初值如下图所示：
```




```
    lfooter=p−1;   rheader=p+[p].size;
    // l表示左邻；r表示右邻；[p].size包括头部和底部
    if (lfooter<lowbound) l=1;
    else l=[lfooter].tag;
    if (rheader>highbound) r=1;
    else r=[rheader].tag;
    return casenum [r] [l];
```

```
} //dealloctype
```

13．试完成伙伴管理系统的存储回收算法。

【解答】对于以下情形，检查算法是否正确：

① 连续合并

② [buddyaddr].tag ==0 && [buddyaddr].kval!=k

③ [buddyaddr].tag ==0 && [buddyaddr].rlink ==buddyaddr

分析问题时要注意以下事实：伙伴块被占用的充要条件：伙伴块一定不在AVAIL表中，故不必查该表，但伙伴地址中标志为0并不一定意味着伙伴块空闲。下面给出算法的核心框架：

```
k=[p].kval; newblock=p; ready=FALSE;
while (k! = m &&! ready) {
    计算伙伴地址buddyaddr;
    // buddyaddr和可能是待插入新块始址的newblock已经准备好。
    if (不能同伙伴合并)
        ready =TRUE;    // newblock即待插入新块始址
    else
    {   // 与伙伴合并
        从空闲表中删除伙伴；
        newblock = min (newblock,  buddyaddr);
        k++;
    }
}
```

将K值为k，始址为newblock的块插入AVAIL表。

14．设被管理空间的上下界地址分别由变量highbound和lowbound给出，形成一个由同样大小的块组成的“堆”。试写一个算法，将所有tag域的值为0的块按始址递增顺序链接成一个可利用空间表（设块大小域为cellsize）。

【分析】赋初值：avail=NULL；p=highbound−cellsize+1；从高地址端向低地址端扫描，while循环的终止条件为(！(p≥lowbound))（在此，假设（highbound−lowbound+1）%cellsize == 0）。

【解答】算法如下：

```
FBList *MakeList(char *highbound，char *lowbound)//把堆结构存储的所有空闲块链接成可利用空间
                                                  //表，并返回表头指针
{
    p=lowbound;
    while(p->tag&&p<highbound) p++;//查找第一个空闲块
    if(p>=highbound) return NULL;//没有空闲块
    head=p;
    for(q=p;p<highbound;p+=cellsize)//建立链表
        if(!p->tag)
        {
            q->next=p;
```

```
            q=p;
        }//if
    p->next=NULL;
    return head;//返回头指针
}//MakeList
```

15．试完成存储紧缩算法。

【解答】算法如下：

```
void Mem_Contract(Heap &H)//对堆H执行存储紧缩
{
    q=MemStart;j=0;
    for(i=0;i<Max_ListLen;i++)
      if(H.list[i].stadr->tag)
      {
          s=H.list[i].length;
          p=H.list[i].stadr;
          for(k=0;k<s;k++) *(q++)=*(p++);//紧缩内存空间
          H.list[j].stadr=q;
          H.list[j].length=s;//紧缩占用空间表
          j++;
      }
}//Mem_Contract
```

8.4　考研真题分析

例题8-1　（清华大学1998年试题）

已知一个大小为512个字节长的存储块，假设先后有6个用户申请大小分别为23，45，52，100，11和19的存储空间，然后再顺序释放大小为45，52和11的占用块。假设以伙伴系统实现动态存储管理。

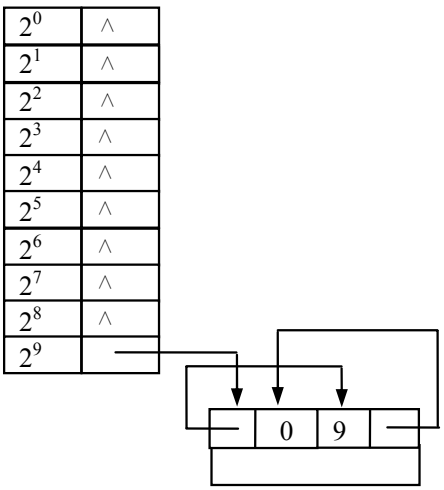
① 画出可利用空间表的初始状态。

② 画出为6个用户分配所需要的存储空间后可利用空间表的状态以及每个用户得到的存储块的起始地址。

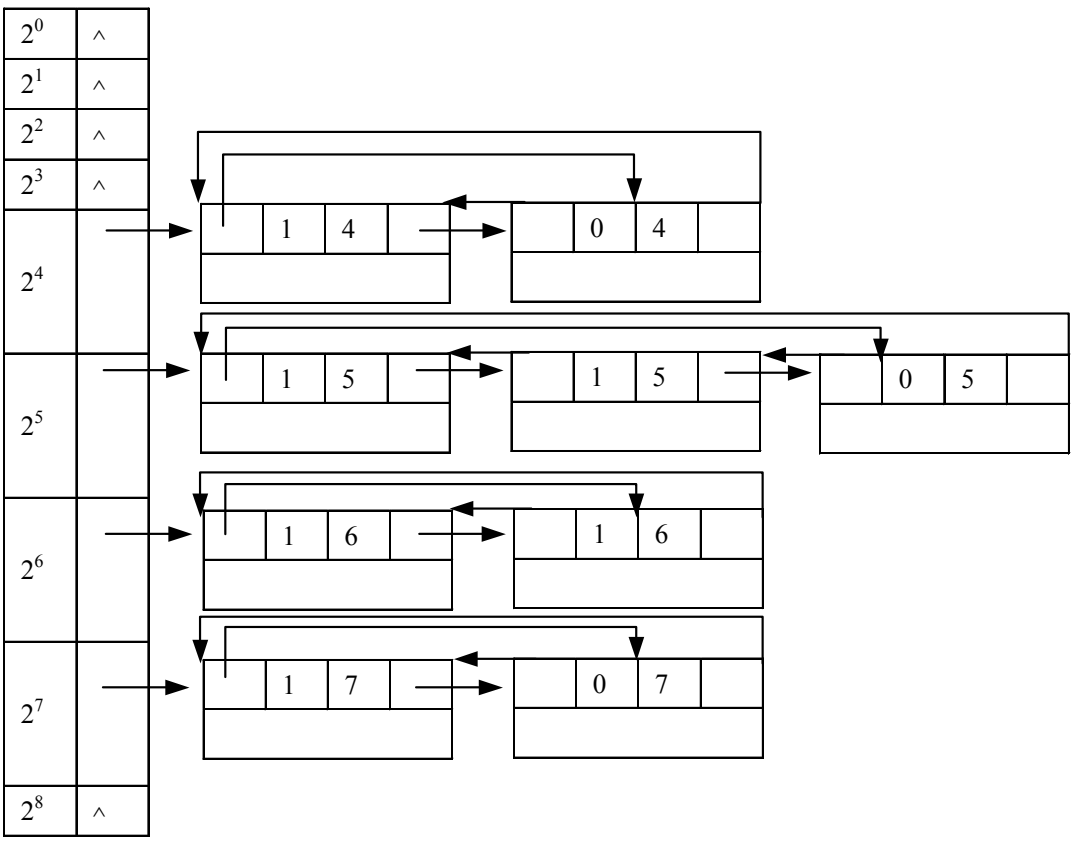
③ 画出在回收3个占用块之后可利用空间表的状态。

【分析】注意伙伴的定义。起始地址为p，大小为2^k的内存块，其伙伴块的起始地址为：$p+2^k$（若pMOD^{2k+1}=0）。否则，其伙伴块的起始地址为：$p-2^k$(若pMOD$2^{k+1}=2^k$)。当进行回收时，即使有两个大小相同并且彼此相邻的空闲块，若它们互不为伙伴，也不能归并到一起。

【解答】① 可利用空间表的初始状态如下图所示。



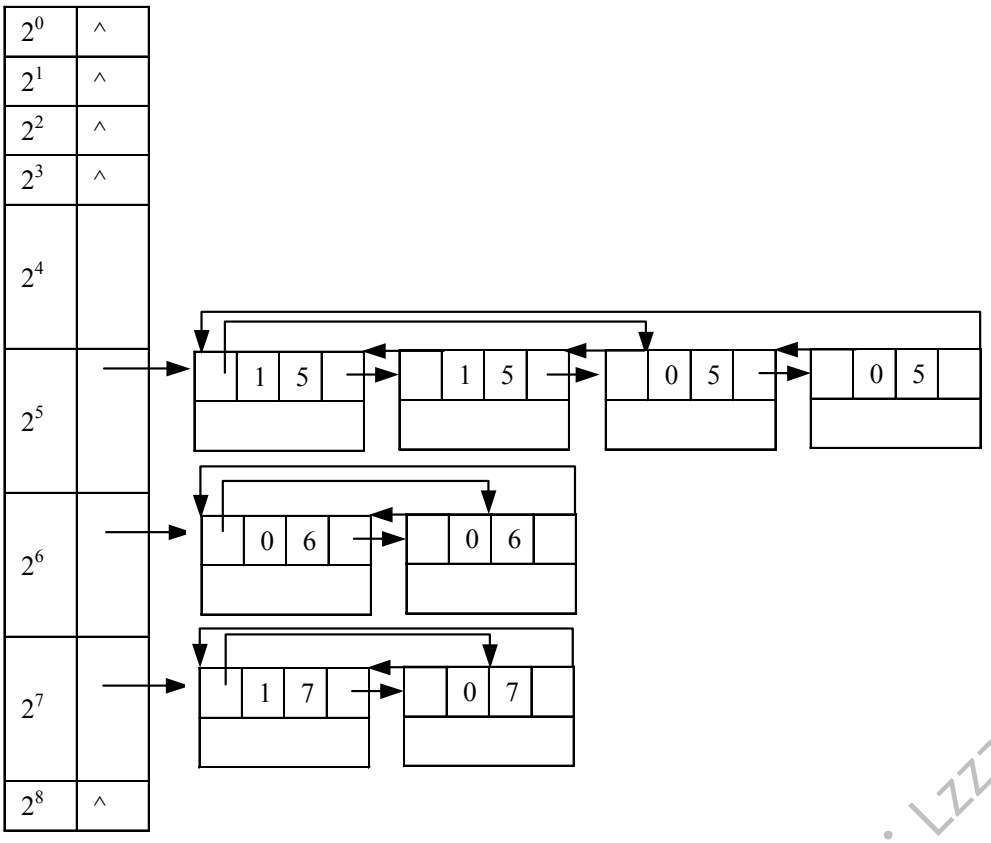
② 为6个用户分配所需要的存储空间后可利用空间表的状态如下图所示。



每个用户得到的存储空间的起始地址见下表。

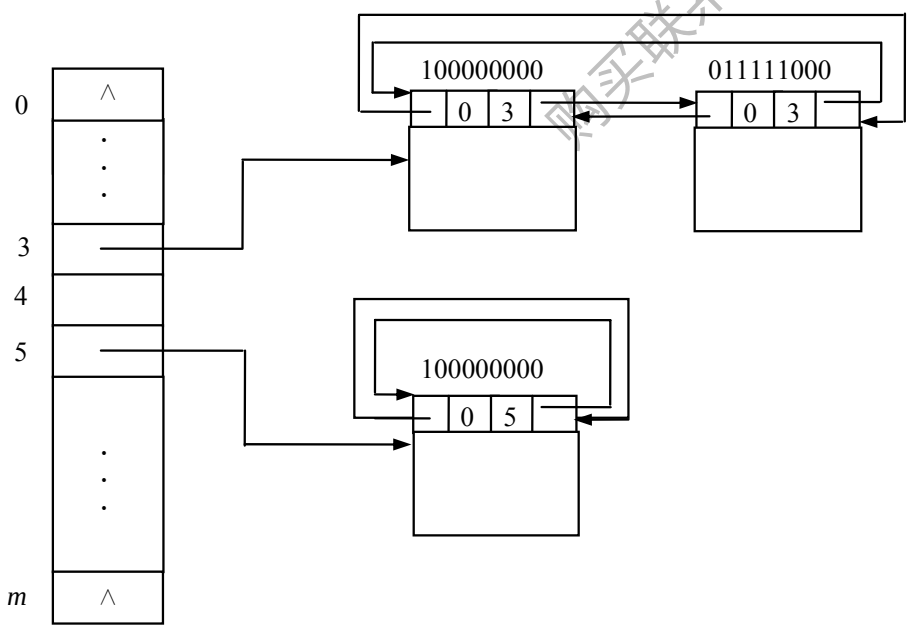
存储大小	起始地址
23	0
45	64
52	128
100	256
11	32
19	192

③ 在回收3个占用块之后可利用空间表的状态如下图所示。



例题8-2 （北京邮电大学1998年试题）

已知伙伴系统的可利用空间表的当前状态如下图所示。



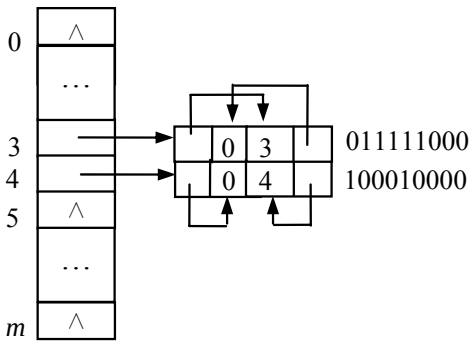
（1）画出分配出去大小为8和16的存储块后，此可利用空间表的状态；

（2）画出在上述分配后再回收首地址为011101000和011110000大小为8的两个存储块后的可利用空间表的状态。

【分析】伙伴系统的分配算法：对于大小为n的内存请求，若恰好有大小为n的内存块，则分配之；否则查找更大的内存块，将一部分分配，剩余部分插入相应子表。

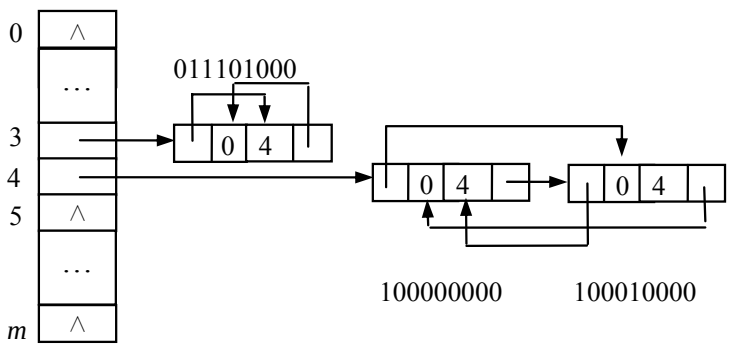
伙伴系统的回收算法：将空闲块插入字表，若与相邻块是伙伴，则合并。

【解答】（1）分配出8和16的存储块如下图所示。



（2）回收首地址为011101000和011110000大小为8的两个存储块后的可利用空间表的状态如下图所示。

011110000和011101000互为伙伴。



8.5 自 测 题

自测题8-1

地址为$(1664)_{10}$，大小为$(128)_{10}$的存储块的伙伴地址是什么？

地址为$(2816)_{10}$，大小为$(64)_{10}$的存储块的伙伴地址是什么？

8.6 自测题答案

【自测题8-1】

起始地址为p，大小为2^k的内存块，其伙伴块的起始地址为

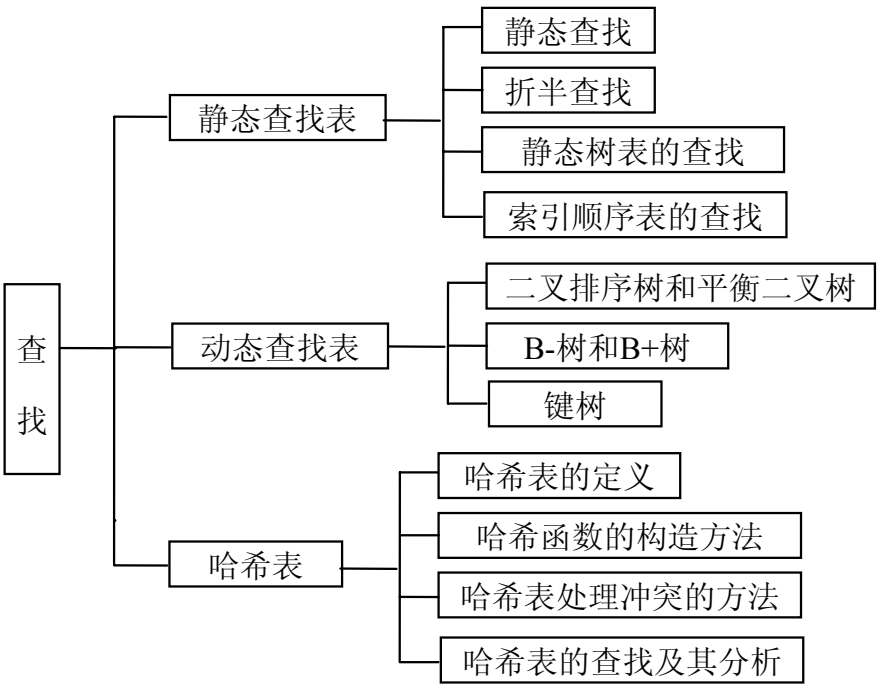
$$\text{buddy}(p，k)=\begin{cases} p+2^k & (\text{若} p \bmod 2^{k+1}=0) \\ p-2^k & (\text{若} p \bmod 2^{k+1}=2^k) \end{cases}$$

答案：

buddy(1664，7)=1664−128=1536
buddy(2816，6)=2816+64=2880

第 9 章　查　找

9.1　基本知识结构图



9.2　知 识 点

1. 查找表

查找表是由同一类型的数据元素（或记录）构成的集合。对查找表经常进行的操作有：（1）查询某个“特定的”数据元素是否在查找表中；（2）检索某个“特定的”数据元素的各种属性；（3）在查找表中插入一个数据元素；（4）从查找表中删除某个数据元素。若对查找表只做前两种统称为“查找”的操作，则称此类查找表为静态查找表；若对查找表同时进行插入或删除操作，则称此类表为动态查找表。

2．静态查找表

以顺序表或线性链表表示的静态查找表可以用顺序查找方法来实现，以有序表表示的静态查找表可用折半查找方法来实现，静态树表的查找可用静态最优查找树和次优查找树

实现，索引顺序表的查找可用分块查找实现。

顺序查找的查找过程为：从表中最后一个记录开始，逐个进行记录的关键字和给定值的比较，若某个记录的关键字和给定值相等，则查找成功，找到所查记录；反之，若直至第一个记录，其关键字和给定值都不等，则表明表中没有所查记录，查找不成功。

折半查找的查找过程为：先确定待查记录所在的范围（区间），然后逐步缩小范围直到找到或找不到该记录为止。

当有序表中各记录的查找概率不等时，可用构造静态最优查找树或次优查找树的方法实现查找，在只考虑查找成功的情况下使查找性能达最佳。

分块查找又称索引顺序查找，是顺序查找的一种改进方法，在此查找法中，除表本身以外，尚需建立一个“索引表”。索引表按关键字有序，则表或者有序或者分块有序。查找时先确定待查记录所在的块，然后在块中顺序查找。

3. 动态查找表

动态查找表的特点是，表结构本身是在查找过程中动态生成的，若表中存在其关键字等于K的记录，则查找成功返回，否则插入关键字等于K的记录。动态查找表可有不同的表示方法：二叉排序树和平衡二叉树、B-树和B+树、键树以及哈希表等。

二叉排序树或者是一棵空树；或者是具有下列性质的二叉树：（1）若它的左子树不空，则左子树上所有结点的值均小于它的根结点的值；（2）若它的右子树不空，则右子树上所有结点的值均大于它的根结点的值；（3）它的左、右子树也分别为二叉排序树。

平衡二叉树又称AVL树，它或者是一颗空树，或者是具有下列性质的二叉树：它的左子树和右子树都是平衡二叉树，且左子树和右子树的深度之差的绝对值不超过1。

B-树是一种平衡的多路查找树，一棵m阶的B-树，或为空树，或为满足下列特征的m叉树：

① 树中每个结点至多有m棵子树；
② 若根结点不是叶子结点，则至少有两棵子树；
③ 除根之外的所有非终端结点至少有$\lceil m/2 \rceil$棵子树；
④ 所有的非终端结点中包含下列信息数据：

（n，A_0，K_1，A_1，K_2，A_2，…，K_n，A_n）

其中，$K_i(i=1, \cdots, n)$为关键字，且$K_i<k_i+1$；A_i为指向子树根结点的指针，且指针A_{i-1}所指子树中所有结点的关键字均小于K_i，A_n所指子树中所有结点的关键字均大于K_n，n为关键字的个数。

⑤ 所有的叶子结点都出现在同一层次上，并且不带信息。

B+树是应文件系统所需而出现的一种B-树的变型树，一棵m阶的B+树和m阶的B-树的差异在于：

① 有m棵子树的结点中含有m个关键字。
② 所有的叶子结点中包含了全部关键字的信息及指向含这些关键字的记录的指针，且叶子结点本身按关键字的大小自小而大顺序链接。
③ 所有的非终端结点可以看成是索引部分，结点中只含有其子树（根结点）中的最大

（或最小）关键字。

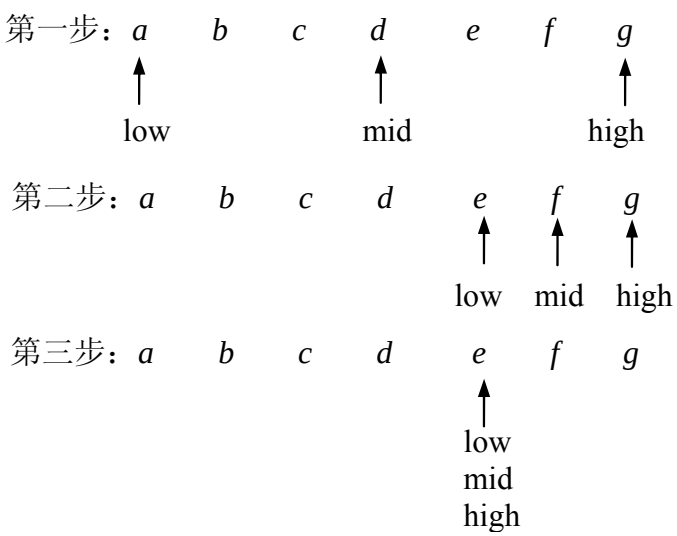
键树又称数字查找树，它是一棵度大于等于2的树，树中的每个结点中不是包含一个或几个关键字，而是只含有组成关键字的符号。

根据设定的哈希函数和处理冲突的方法将一组关键字映像到一个有限的连续的地址集（区间）上，并以关键字在地址集中的“像”作为记录在表中的存储位置，这种表便称为哈希表，这一映像过程称为哈希造表或散列，所得存储位置成为哈希地址或散列地址。构造哈希函数的方法有：直接定址法、数字分析法、平方取中法、折叠法、除留余数法和随机数法六种；哈希表处理冲突的方法有：开放定址法、再哈希法、链地址法和建立一个公共溢出区的方法。

9.3 习题及参考答案

1．试分别画出在线性表（*a*，*b*，*c*，*d*，*e*，*f*，*g*）中进行折半查找，以查关键字等于*e*、*f*和*g*的过程。

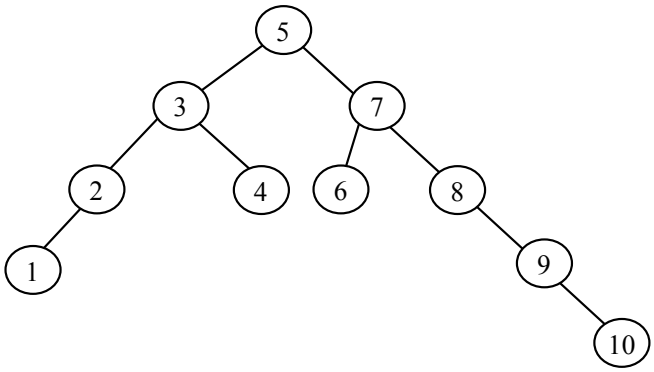
【解答】关键字等于*e*的查找过程（其他从略）如下：



查找成功。

2．画出对长度为10的有序表进行折半查找的判定树，并求其等概率时查找成功的平均查找长度。

【解答】判定树如下所示：

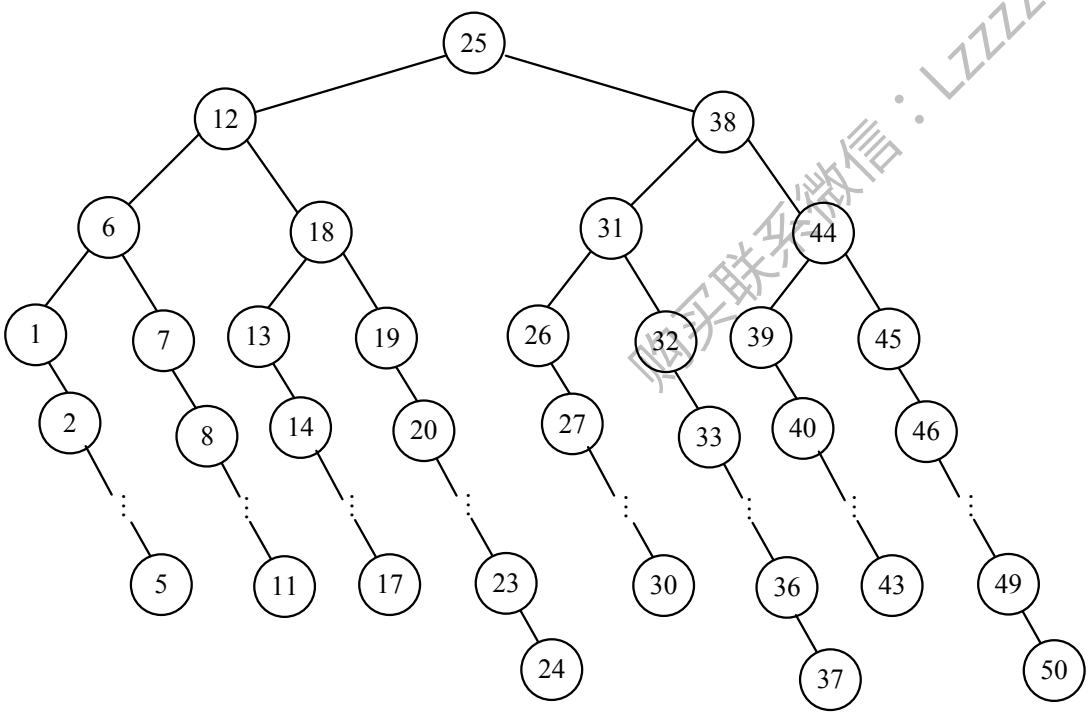


等概率查找时成功的平均查找长度为

$$ASL_{succ}=\frac{1}{10}(1\times1+2\times2+3\times4+4\times3)=2.9$$

3．假设按下述递归方法进行顺序表的查找：若表长小于等于10，则进行顺序查找，否则进行折半查找。试画出对表长n=50的顺序表进行上述查找时，描述该查找的判定树，并求出在等概率情况下查找成功的平均查找长度。

【解答】判定树如下：



其等概率查找时查找成功的平均查找长度为

$$ASL_{succ}=\frac{1}{50}(1\times1+2\times2+3\times4+(4+5+6+7+8)\times8+9\times3)=5.68$$

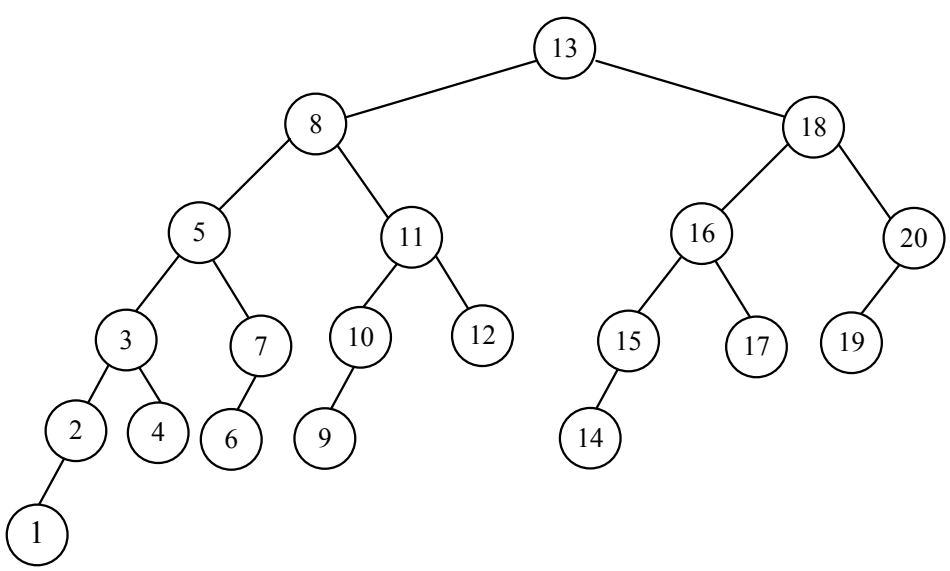
4．下列算法为斐波那契查找算法：

```
int FibSearch (SqList r，  KeyType K) {
      j=1;
```

```
    while (fit(j)<n+1) j=j+1;
    mid=n−fib(j−2)+1;   //若fib(j)=n+1 则mid=fib(j−1)
    f1=fib(j−2);   f2=fib(j−3);   found=FALSE;
    while ((mid<>0) && !found)
        switch{
            case K=r[mid].key;   found=TRUE; break;
            case K<r[mid].key;
                if (!f2) mid=0;
                else {mid=mid−f2;   t=f1−f2;   f1=f2;   f2=t;}
                break;
            case K>r[mid].key;
                if (f1=1) mid=0;
                else {mid=mid+f2;   f1=f1−f2;   f2=f2−f1;}
                break;
            }
    if (found) return mid;
    else return 0;
}   //FibSearch
```

其中fib(i)为斐波那契序列。试画出对长度为20的有序表进行斐波那契查找的判定树，并求在等概率时查找成功的平均查找长度。

【解答】判定树如下：



其等概率查找时查找成功的平均查找长度为

$$ASL_{succ}=\frac{1}{20}(1\times1+2\times2+3\times4+4\times7+5\times5+6\times1)=3.8$$

5．假设在某程序中有如下if…then嵌套语句：

```
if (C1)
    if (C2)
        if (C3)
            …
                if (Cn)
                    S;
```

其中C_i为布尔表达式。显然，只有当所有的C_i都为TRUE时，语句S才能执行。假设$t(i)$为判别C_i是否为TRUE所需时间，$p(i)$为C_i是TRUE的概率，试讨论这n个布尔表达式C_i（i=1，2，…，n）应如何排列才能使该程序最有效地执行。

【解答】应该先列出该语句执行时间的期望值。

假设语句S的执行时间为T，则该语句执行时间的期望值为

$$T=(1-p(1))\,t(1)+p(1)(1-p(2))(t(1)+t(2))+\cdots$$
$$+p(1)p(2)\cdots p(n-1)(1-p(n))(t(1)+t(2)+\cdots$$
$$+t(n))+p(1)p(2)+\cdots p(n)(t(1)+t(2)\cdots+t(n)+t)$$

令$p(0)=1$，则上式可写为

$$T=(\prod_{j=1}^{n}p(j))t+\sum_{j=1}^{n}t(j)(\sum_{i=1}^{n}(\prod_{j=0}^{i-1}p(j))(1-p(i))+\prod_{j=1}^{n}p(j))$$
$$=(\prod_{j=1}^{n}p(j))t+\sum_{i=1}^{n}(\prod_{j=0}^{i-1}p(j))t(i)$$

可以证明，当这n个布尔表达式C_i的排列满足下式时，将使T达极小值：

$$\frac{t(1)}{1-p(1)}<\frac{t(2)}{1-p(2)}<\cdots<\frac{t(n)}{1-p(n)}$$

因此，应按$\dfrac{t(i)}{1-p(i)}$从小至大的次序来排列，将使该语句的执行最有效。

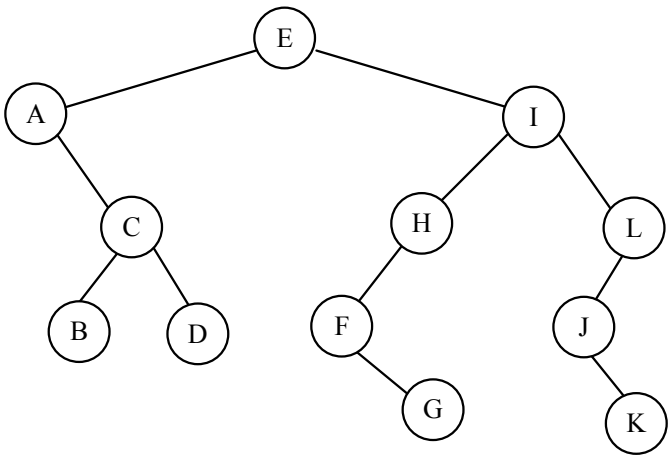
6．已知含12个关键字的有序表及其相应权值如下所示：

关键字	A	B	C	D	E	F	G	H	I	J	K	L
权值	8	2	3	4	9	3	2	6	7	1	1	4

（1）试按次优查找树的构造算法并加适当调整画出由这12个关键字构造所得的次优查找树，并计算它的PH值；

（2）画出对以上有序表进行折半查找的判定树，并计算它的PH值。

【解答】（1）次优查找树如下所示，其PH值为133；



（2）折半查找的判定树的*PH*值为156。

7．已知如下所示长度为12的表

（Jan，Feb，Mar，Apr，May，June，July，Aug，Sep，Oct，Nov，Dec）

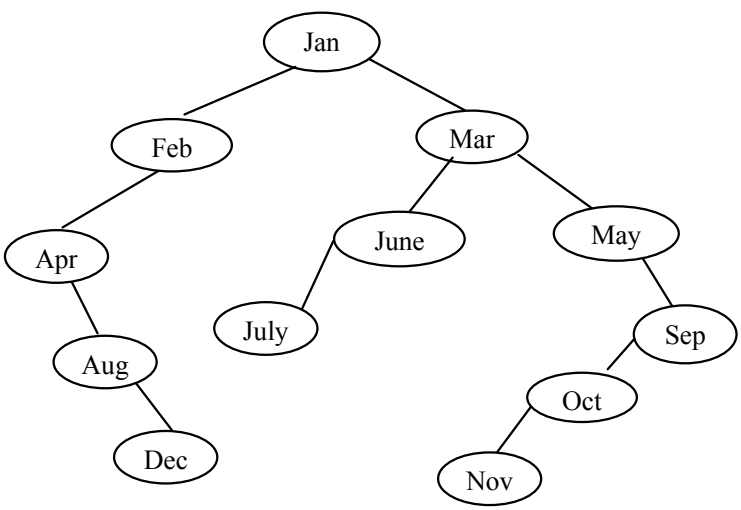
（1）试按表中元素的顺序依次插入一棵初始为空的二叉排序树，画出插入完成之后的二叉排序树，并求其在等概率的情况下查找成功的平均查找长度。

（2）若对表中元素先进行排序构成有序表，求在等概率的情况下对此有序表进行折半查找时查找成功的平均查找长度。

（3）按表中元素顺序构造一棵平衡二叉排序树，并求其在等概率的情况下查找成功的平均查找长度。

【解答】（1）求得的二叉排序树如下图所示，在等概率情况下查找成功的平均查找长度为

$$ASL_{succ}=\frac{1}{12}(1\times1+2\times2+3\times3+4\times3+5\times2+6\times1)=\frac{42}{12}$$



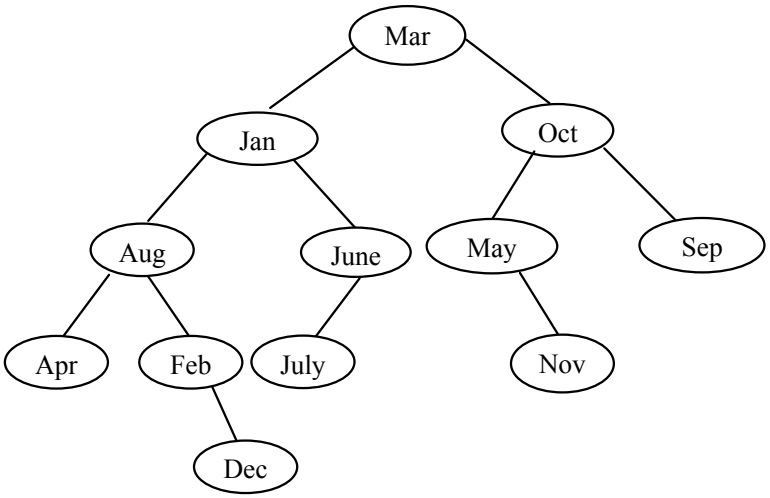
（2）经排序后的表及在折半查找时找到表中元素所经比较的次数对照如下：

Apr	Aug	Dec	Feb	Jan	July	June	Mar	May	Nov	Oct	Sep
3	4	2	3	4	1	3	4	2	4	3	4

等概率情况下查找成功时的平均查找长度为

$$ASL_{succ}=\frac{1}{12}(1\times1+2\times2+3\times4+4\times5)=\frac{37}{12}$$

（3）平衡二叉树如下：

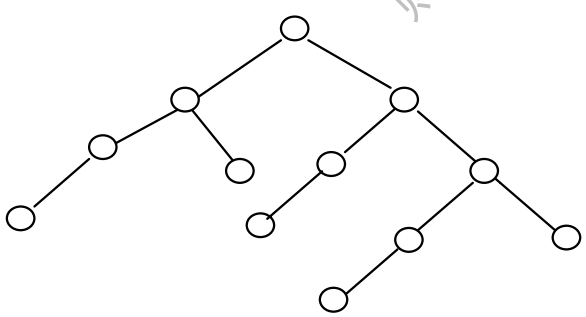


它在等概率情况下的平均查找长度为

$$ASL_{succ}=\frac{1}{12}(1\times1+2\times2+3\times4+4\times4+5\times1)=\frac{38}{12}$$

8．试推导含12个结点的平衡二叉树的最大深度，并画出一棵这样的树。

【解答】深度为h的平衡二叉树中含有的最少结点为$N_h=N_{h-1}+N_{h-2}+1$，由此可推出含12个结点的平衡树的最大深度为5。

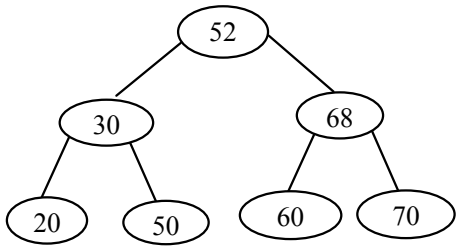


9．含9个叶子结点的3阶B-树中至少有多少个非叶子结点？含10个叶子结点的3阶B-树中至多有多少个非叶子结点？

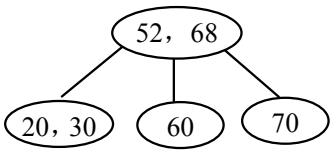
【解答】至少含4个非叶子结点；至多含8个非叶子结点

10．试从空树开始，画出按以下次序向2-3树即3阶B-树中插入关键码的建树过程：20，30，50，52，60，68，70。如果此后删除50和68，画出执行每一步后2-3树的状态。

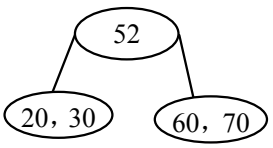
【解答】建成的树如下：



删去50之后的树如下：



再删去68之后的树如下：



11．在含有n个关键码的m阶B−树中进行查找时，最多访问多少个结点？

【解答】$\log_{\lceil m/2 \rceil}(\frac{n+1}{2})+1$

12．已知一个含有1000个记录的表，关键字为中国人姓氏的拼音，请给出此表的一个哈希表设计方案，要求它在等概率情况下查找成功的平均查找长度不超过3。

【解答】设计哈希表的步骤为：

（1）根据所选择的处理冲突的方法求出装载因子α的上界；

（2）由α值设计哈希表的长度m；

（3）根据关键字的特性和表长m选定合适的哈希函数。

13．设有一个关键字取值范围为正整数的哈希表，空表项的值为−1，用开放定址法解决冲突。现有两种删除策略：一是将待删表项的关键字置为−1；二是将探测序列上的关键字顺序递补，即用探测序列上下一个关键字覆盖待删关键字，并将原序列上之后一个关键字置为−1。这两种方法是否可行？为什么？给出一种可行的方法，并叙述它对查找和插入算法所产生的影响。

【解答】两种策略都不可行。前者切断了探测链；后者可能移动了非同义词。一种可行的做法是将待删表项的关键字置为0，以区别于空表项。查找和插入算法都应相应地进行调整。

14．某校学生学号由8位十进制数字组成：$C_1\,C_2\,C_3\,C_4\,C_5\,C_6\,C_7\,C_8$。$C_1\,C_2$为入学时年份的后两位。$C_3\,C_4$为系别：00~24分别代表该校的25个系；$C_5$为0或1，0表示本科生，1表示研究生。$C_6\,C_7\,C_8$为对某级某系某类学生的顺序编号：对于本科生，它不超过199；对于研究生，它不超过049；共有4个年级，四年级学生1996年入学。

（1）当在校生人数达极限情况时，将他们的学号散列到0~24999的地址空间，问装载因子是多少？

（2）求一个无冲突的哈希函数H_1，它将在校生学号散列到0~24999的地址空间。其簇聚性如何？

（3）设在校生总数为15000人，散列地址空间为0~19999，你是否能找到一个（2）中要求的H_1？若不能，试设计一个哈希函数H_2及其解决冲突的方法，使得多数学号可只经一次散列得到（可设各系各年级本科生平均人数为130，研究生平均人数为20）。

【解答】设计哈希表时，若有可能利用直接定址找到关键字和地址一一对应的映像函数，则是大好事。本题旨在使读者认识到，复杂关键字集合或要求的装载因子α接近1时，不一定找不到一一对应的映像函数。

（1）α=1;

（2）这样的哈希函数不惟一，一种方案是

$$H_1(C)=C_{678}+C_5\times 200+C_{34}\times(200+50)+(C_{12}-96)\times((200+50)\times 25)$$

其中，$C=C_1C_2C_3C_4C_5C_6C_7C_8$

$C_{678}=C_6\times 100+C_7\times 10+C_8$

$C_{34}=C_3\times 10+C_4$

$C_{12}=C_1\times 10+C_2$

此哈希函数在无冲突的前提下非常集聚。

（3）不能（虽然α<1）。哈希函数可设计为

$$H_2(C)=(1-2C_5)C_{678}+C_5(l-1)+C_{34}l+(C_{12}-96)\times 25l$$

其中，$C_{678}=C_6\times 100+C_7\times 10+C_8$，其余类推。

方案一：l=150，最后5000个表项作为公共溢出区；

方案二：l=200，用开放定址处理冲突。也可有其他折衷方案。

15．假设顺序表按关键字自大到小有序，试改写顺序查找算法，将监视哨设在高下标端。分别求出等概率情况下查找成功和不成功时的平均查找长度。

【解答】算法如下：

```
int Search_Sq(SSTable ST，int key)//在有序表上顺序查找的算法，监视哨设在高下标端
{
    ST.elem[ST.length+1].key=key;
    for(i=1;ST.elem[i].key>key;i++);
    if(i>ST.length| |ST.elem[i].key<key) return ERROR;
    return i;
}//Search_Sq
```

本算法查找成功情况下的平均查找长度为ST.length/2，不成功情况下为ST.length。

16. 试将折半查找的算法改写成递归算法。

【解答】折半查找递归调用的算法如下：

```
int BinSearch (SSTable s; int low，  int high; keyType K)
{
// 在顺序表s的s.elem[low..high]上进行折半查找，K为给定值，查找成功
// 时，返回的函数值为关键字等于给定值的记录在顺序表中的位置（序号）
if (low>high) return 0;        // 查找不成功
```

```
else {
        mid = (low+high) /2;
        switch {
                case   s.elem [mid].key<K:
                        return BinSearch (s，  mid+1，  high，  K);
                        break;
                case s.elem [mid].key == K:
                        return mid;
                        break;
                case s.elem [mid].key>K:
                        return BinSearch (s，  low，  mid−1，  K);
                        break;
            default: ;
        }
} //BinSearch
}
```

17．试编写利用折半查找确定记录所在块的分块查找算法。并讨论在块中进行顺序查找时使用“监视哨”的优缺点，以及必要时如何在分块查找的算法中实现设置“监视哨”的技巧。

【解答】本题所用的存储结构，除了顺序表data[DataNum]外，还建有索引表idxtab[MaxIndex]，其中idxtab[MaxBlk]内含有各块索引。

```
typedef struct {   // 索引项定义
        KeyType max key;     // 各块中最大关键字
        indexType idx;          // 各块的初始序号
                        // 若每次大小不等，则还要加一个keynum域
} IndexItem;
typedef struct {   // 索引顺序表定义
        ElemType *data;
        IndexItem *idxtab;
        int MaxBlk;
        int BlockSize;
} IndexSqlist;
```

设每块大小相同，都为BlockSize，且除最后一块外的每块都装满。

在索引表上进行折半查找的策略是找一个“缝隙”。即求i满足idxtab[0..i−1].key<K≤idxtab[i..MaxBlk−1].key (0≤i≤MaxBlk)。

在块中进行顺序查找时，监视哨可设在本块的表尾，即将下一块的第一个记录暂时移走（若本块内记录没有填满，则监视哨的位置仍在本块的尾部），待块内顺序查找完成以后再移回来。此时增加了赋值运算，但免去了判断下标变量是否出界的比较。注意最后一块尚需进行特殊处理。

注意算法在边界条件下执行的正确性：

① K>idxtab[MaxBlk−1].key;

② K=data[DataNum−1].key。

18．已知一非空有序表，表中记录按关键字递增排列，以不带头结点的单循环链表作

存储结构，外设两个指针h和t，其中h始终指向关键字最小的结点，t则在表中浮动，其初始位置和b相同，在每次查找之后指向刚查到的结点。查找算法的策略是：首先将给定值K和t->key进行比较，若相等，则查找成功；否则因K小于或大于t->key而从h所指结点或t所指结点的后继结点进行查找。

（1）按上述查找过程编写查找算法；

（2）分析在等概率查找时查找成功的平均查找长度（假设表长为n，待查关键码K等于每个结点关键码的概率为1/n，每次查找都是成功的，因此在查找时，t指向每个结点的概率也为1/n）。

【解答】（1）算法如下：

```
typedef struct {
                    LNode *h; //h指向最小元素
                    LNode *t; //t指向上次查找的结点
                } CSList;
LNode *Search_CSList(CSList &L，int key)//在有序单循环链表存储结构上的查找算法，假定每次查
                                        //找都成功
{
    if(L.t->data==key) return L.t;
    else if(L.t->data>key)
          for(p=L.h，i=1;p->data!=key;p=p->next，i++);
     else
          for(p=L.t，i=L.tpos;p->data!=key;p=p->next，i++);
    L.t=p; //更新t指针
    return p;
}//Search_CSList
```

（2）由于题目中假定每次查找都是成功的，所以本算法中没有关于查找失败的处理。由微积分可得，在等概率情况下，平均查找长度约为n/3。

19．将18题的存储结构改为双向循环链表，且只外设一个指针sp，其初始位置指向关键字最小的结点，在每次查找之后指向刚查到的结点。查找算法的策略是：首先将给定值K和sp->key进行比较，若相等，则查找成功；否则依K小于或大于sp->key继续从*sp的前驱或后继结点起进行查找。编写查找算法并分析等概率查找时查找成功的平均查找长度。

【解答】算法如下：

```
typedef struct {
                    DLNode *pre;
                    int data;
                    DLNode *next;
                } DLNode;
typedef struct {
                     DLNode *sp;
                     int length;
                 } DSList; //供查找的双向循环链表类型
DLNode *Search_DSList(DSList &L，int key)//在有序双向循环链表存储结构上的查找算法，假定每次
                                         //查找都成功
```

```
{
    p=L.sp;
    if(p->data>key)
    {
        while(p->data>key) p=p->pre;
        L.sp=p;
    }
    else if(p->data<key)
        {
            while(p->data<key) p=p->next;
            L.sp=p;
        }
    return p;
}//Search_DSList
```

本题的平均查找长度与上一题相同，也是$n/3$。

20．试写一个判别给定二叉树是否为二叉排序树的算法，设此二叉树以二叉链表作存储结构。且树中结点的关键字均不同。

【分析】注意仔细研究二叉排序树的定义。易犯的典型错误是按下述思路进行判别：“若一棵非空的二叉树其左、右子树均为二叉排序树，且左子树的根的值小于根结点的值，又根结点的值不大于右子树的根的值，则是二叉排序树”。

【解答】算法如下：

```
int last=0，flag=1;
int Is_BSTree(Bitree T)//判断二叉树T是否二叉排序树，是则返回1，否则返回0
{
    if(T->lchild&&flag) Is_BSTree(T->lchild);
    if(T->data<last) flag=0; //与其中序前驱相比较
    last=T->data;
    if(T->rchild&&flag) Is_BSTree(T->rchild);
    return flag;
}//Is_BSTree
```

21．已知一棵二叉排序树上所有关键字中的最小值为$-max$，最大值为max，又$-max<x<max$。编写递归算法，求该二叉排序树上的小于x且最靠近x的值a和大于x且最靠近x的值b。

【解答】算法如下：

```
int last=0;
void MaxLT_MinGT(BiTree T，int x)//找到二叉排序树T中小于x的最大元素和大于x的最小元素
{
    if(T->lchild) MaxLT_MinGT(T->lchild，x); //本算法仍是借助中序遍历来实现
    if(last<x&&T->data>=x) //找到了小于x的最大元素
        printf("a=%d\n"，last);
    if(last<=x&&T->data>x) //找到了大于x的最小元素
        printf("b=%d\n"，T->data);
    last=T->data;
    if(T->rchild) MaxLT_MinGT(T->rchild，x);
```

```
}//MaxLT_MinGT
```

22．编写递归算法，从大到小输出给定二叉排序树中所有关键字不小于x的数据元素。要求你的算法的时间复杂度为$O(\log_2 n+m)$，其中n为排序树中所含结点数，m为输出的关键字个数。

【分析】为满足题目所要求的时间复杂度，算法中应注意做到，一旦访问到关键字小于x的结点，立即结束遍历。

【解答】算法如下：

```
void Print_NLT(BiTree T，int x)//从大到小输出二叉排序树T中所有不小于x的元素
{
    if(T->rchild) Print_NLT(T->rchild，x);
    if(T->data<x) exit(); //当遇到小于x的元素时立即结束运行
    printf("%d\n"，T->data);
    if(T->lchild) Print_NLT(T->lchild，x); //先右后左的中序遍历
}//Print_NLT
```

23．假设二叉排序树以后继线索链表作存储结构，编写输出该二叉排序树中所有大于a且小于b的关键字的算法。

【分析】虽然在题目中没有指出本题的存储结构是哪一种“序”的线索链表，但很明显，对于二叉排序树而言，只能是“中序”线索链表。

【解答】算法如下：

```
void Print_Between(BiThrTree T，int a，int b)//打印输出后继线索二叉排序树T中所有大于a且小于b的
                                              //元素
{
    p=T;
    while(!p->ltag) p=p->lchild; //找到最小元素
    while(p&&p->data<b)
    {
        if(p->data>a) printf("%d\n"，p->data); //输出符合条件的元素
        if(p->rtag) p=p->rtag;
        else
        {
            p=p->rchild;
            while(!p->ltag) p=p->lchild;
        } //转到中序后继
    }//while
}//Print_Between
```

24. 同23题的结构，编写在二叉排序树中插入一个关键字的算法。

【解答】算法如下：

```
void BSTree_Insert_Key(BiThrTree &T，int x)//在后继线索二叉排序树T中插入元素x
{
    if(T->data<x) //插入到右侧
    {
        if(T->rtag) //T没有右子树时，作为右孩子插入
```

```
        {
            p=T->rchild;
            q=(BiThrNode*)malloc(sizeof(BiThrNode));
            q->data=x;
            T->rchild=q;T->rtag=0;
            q->rtag=1;q->rchild=p; //修改原线索
        }
        else BSTree_Insert_Key(T->rchild，x);//T有右子树时，插入右子树中
    }//if
    else if(T->data>x) //插入到左子树中
    {
        if(!T->lchild) //T没有左子树时，作为左孩子插入
        {
            q=(BiThrNode*)malloc(sizeof(BiThrNode));
            q->data=x;
            T->lchild=q;
            q->rtag=1;q->rchild=T; //修改自身的线索
        }
        else BSTree_Insert_Key(T->lchild，x);//T有左子树时，插入左子树中
     }//if
}//BSTree_Insert_Key
```

25．同23题的结构，编写从二叉排序树中删除一个关键字的算法。

【分析】在下面的算法中采用了先求出关键字*x*结点的前驱和后继，再删除*x*结点的办法，这样修改线索时会比较简单，直接让前驱的线索指向后继就行了，如果试图在删除*x*结点的同时修改线索，则问题反而复杂化了。

【解答】算法如下：

```
Status BSTree_Delete_key(BiThrTree &T，int x)//在后继线索二叉排序树T中删除元素x
{
  BTNode *pre，*ptr，*suc;//ptr为x所在结点，pre和suc分别指向ptr的前驱和后继
  p=T;last=NULL; //last始终指向当前结点p的前一个（前驱）
  while(!p->ltag) p=p->lchild; //找到中序起始元素
  while(p)
  {
    if(p->data==x) //找到了元素x结点
    {
      pre=last;
      ptr=p;
    }
    else if(last&&last->data==x) suc=p; //找到了x的后继
    if(p->rtag) p=p->rtag;
    else
    {
      p=p->rchild;
      while(!p->ltag) p=p->lchild;
    } //转到中序后继
    last=p;
  }//while //借助中序遍历找到元素x及其前驱和后继结点
```

```
    if(!ptr) return ERROR; //未找到待删结点
    Delete_BSTree(ptr); //删除x结点
    if(pre&&pre->rtag)
      pre->rchild=suc; //修改线索
    return OK;
}//BSTree_Delete_key
void Delete_BSTree(BiThrTree &T)//删除二叉排序树的子树T的算法，按照线索二叉树的结构作了一
                                //些改动
{
    q=T;
    if(!T->ltag&&T->rtag) //结点无右子树，此时只需重接其左子树
      T=T->lchild;
    else if(T->ltag&&!T->rtag) //结点无左子树，此时只需重接其右子树
            T=T->rchild;
         else if(!T->ltag&&!T->rtag) //结点既有左子树又有右子树
              {
                  p=T;r=T->lchild;
                  while(!r->rtag)
                  {
                      s=r;
                      r=r->rchild; //找到结点的前驱r和r的双亲s
                  }
                  T->data=r->data; //用r代替T结点
                  if(s!=T) s->rchild=r->lchild;
                  else s->lchild=r->lchild; //重接r的左子树到其双亲结点上
                  q=r;
            }//else
      free(q); //删除结点
}//Delete_BSTree
```

26．试写一算法，将两棵二叉排序树合并为一棵二叉排序树。

【分析】在合并过程中，并不释放或新建任何结点，而是采取修改指针的方式来完成合并，这样，就必须按照后序序列把一棵树中的元素逐个连接到另一棵树上，否则将会导致树的结构的混乱。

【解答】算法如下：

```
void BSTree_Merge(BiTree &T，BiTree &S)//把二叉排序树S合并到T中
{
    if(S->lchild) BSTree_Merge(T，S->lchild);
    if(S->rchild) BSTree_Merge(T，S->rchild); //合并子树
    Insert_Key(T，S); //插入元素
}//BSTree_Merge
void Insert_Node(Bitree &T，BTNode *S)//把树结点S插入到T的合适位置上
{
    if(S->data>T->data)
    {
        if(!T->rchild) T->rchild=S;
        else Insert_Node(T->rchild，S);
    }
```

```
    else if(S->data<T->data)
    {
        if(!T->lchild) T->lchild=S;
        else Insert_Node(T->lchild，S);
    }
    S->lchild=NULL; //插入的新结点必须和原来的左右子树断绝关系
    S->rchild=NULL; //否则会导致树结构的混乱
}//Insert_Node
```

27．试写一算法，将一棵二叉排序树分裂为两棵二叉排序树，使得其中一棵树的所有结点的关键字都小于或等于x，另一棵树的任一结点的关键字均大于x。

【分析】以x为分界点遍历原二叉树并构造两棵新的二叉排序树。

【解答】算法如下：

```
void BSTree_Split(BiTree &T，BiTree &A，BiTree &B，int x)
  //把二叉排序树T分裂为两棵二叉排序树A和B，其中A的元素全部小于等于x，B的元素全部大于x
{
    if(T->lchild) BSTree_Split(T->lchild，A，B，x);
    if(T->rchild) BSTree_Split(T->rchild，A，B，x); //分裂左右子树
    if(T->data<=x) Insert_Node(A，T);
    else Insert_Node(B，T); //将元素结点插入合适的树中
}//BSTree_Split
void Insert_Node(Bitree &T，BTNode *S)//把树结点S插入到T的合适位置上
{
    if(!T) T=S; //考虑到刚开始分裂时树A和树B为空的情况
    else if(S->data>T->data) //其余部分与上一题同
    {
        if(!T->rchild) T->rchild=S;
        else Insert_Node(T->rchild，S);
    }
    else if(S->data<T->data)
    {
        if(!T->lchild) T->lchild=S;
        else Insert_Node(T->lchild，S);
    }
    S->lchild=NULL;
    S->rchild=NULL;
}//Insert_Key
```

28．在平衡二叉排序树的每个结点中增设一个lsize域，其值为它的左子树中的结点数加1。试写一时间复杂度为$O(\log n)$的算法，确定树中第k小的结点的位置。

【解答】容易看出，结点中增设的lsize域的值即为该结点在排序树上的“次序”。

```
typedef struct {
                int data;
                int bf;
                int lsize; //lsize域表示该结点的左子树的结点总数加1
                BlcNode *lchild，*rchild;
            } BlcNode，*BlcTree; //含lsize域的平衡二叉排序树类型
```

```
BTNode *Locate_BlcTree(BlcTree T，int k)//在含lsize域的平衡二叉排序树T中确定第k小的结点指针
{
    if(!T) return NULL; //k小于1或大于树结点总数
    if(T->lsize==k) return T; //就是这个结点
    else if(T->lsize>k)
        return Locate_BlcTree(T->lchild，k); //在左子树中寻找
     else return Locate_BlcTree(T->rchild，k-T->lsize); //在右子树中寻找，注意要修改k的值
}//Locate_BlcTree
```

29．为B+树设计结点类型并写出算法，随机（而不是顺序）地查找给定的关键字K，求得它所在的叶子结点指针和它在该结点中位置（提示：B+树中有两种类型的指针，结点结构也不尽相同，考虑利用记录的实体）。

【解答】B+树的查找算法与分块查找算法十分类似。实际上，B+树就是更发达的分块索引数据结构。

B+树的结点类型可设计为

```
typedef enum {LEAF，  NONLEAF} NodeType;
typedef struct BplusNode {
    NodeType      tag;
    int           keynum;
    BPlusLink     parent;
    KeyType       *key;
    union {
        BPlusLink   son [m];      //tag == NONLEAF
        struct { BplusNode   *next;
            reclink     info[m];    //reclink为指向记录的指针
        } leaf;    //tag == LEAF
    }
} BplusNode，    *BPlusLink;
typedef struct
    bool      found;
    BPlusLink leafptr;
    int   position;
} Result，   *ResultPtr;
ResultPtr search (BplusLink root; KeyType K)
{
    // 若查找成功，found=TRUE，p->info[i]指信息，否则found=FALSE，
    // K应在的位置为p->key[i-1]与p->key[i]之间，其中p指K所在或应
    // 在的叶子结点。
}
```

30．假设Trie树上叶子结点的最大层次为*h*，同义词放在同一叶子结点中，试写在Trie树中插入一个关键字的算法。

【分析】当自上而下的查找结束时，存在两种情况：一种情况是树中没有待插入关键字的同义词，此时只要新建一个叶子结点并连到分支结点上即可；另一种情况是有同义词，此时要把同义词的叶子结点与树断开，在断开的部位新建一个下一层的分支结点，再把同义词和新关键字的叶子结点连到新分支结点的下一层。

【解答】算法如下：

```
void TrieTree_Insert_Key(TrieTree &T，StringType key)//在Trie树T中插入字符串key
{
    q=(TrieNode*)malloc(sizeof(TrieNode));
    q->kind=LEAF;
    q->lf.k=key; //建叶子结点
    klen=key[0];
    p=T;i=1;
    while(p&&i<=klen&&p->bh.ptr[ord(key[i])])
    {
        last=p;
        p=p->bh.ptr[ord(key[i])];
        i++;
    } //自上而下查找
    if(p->kind==BRANCH) //如果最后落到分支结点(无同义词):
    {
        p->bh.ptr[ord(key[i])]=q; //直接连上叶子
        p->bh.num++;
    }
    else //如果最后落到叶子结点(有同义词):
    {
        r=(TrieNode*)malloc(sizeof(TrieNode)); //建立新的分支结点
        last->bh.ptr[ord(key[i-1])]=r; //用新分支结点取代老叶子结点和上一层的联系
        r->kind=BRANCH;r->bh.num=2;
        r->bh.ptr[ord(key[i])]=q;
        r->bh.ptr[ord(p->lf.k[i])]=p; //新分支结点与新老两个叶子结点相连
    }
}//TrieTree_Insert_Key
```

31．同30题的假设，试写在Trie树中删除一个关键字的算法。

【解答】算法如下：

```
Status TrieTree_Delete_Key(TrieTree &T，StringType key)//在Trie树T中删除字符串key
{
    p=T;i=1;
    while(p&&p->kind==BRANCH&&i<=key[0]) //查找待删除元素
    {
        last=p;
        p=p->bh.ptr[ord(key[i])];
        i++;
    }
    if(p&&p->kind==LEAF&&p->lf.k=key) //找到了待删除元素
    {
        last->bh.ptr[ord(key[i-1])]=NULL;
        free(p);
        return OK;
    }
    else return ERROR; //没找到待删除元素
}//TrieTree_Delete_Key
```

32．已知某哈希表的装载因子小于1，哈希函数*H*(*key*)为关键字（标识符）的第一个字母在字母表中的序号，处理冲突的方法为线性探测开放定址法。试编写一个按第一个字母的顺序输出哈希表中所有关键字的算法。

【分析】注意此题给出的条件：装载因子α<1，则哈希表未填满。由此可写出下列形式简明的算法。

【解答】算法如下：

```
void PrintWord (HashTable ht)
{
      // 按第一个字母的顺序输出哈希表ht中的标识符。哈希函数为标识符的
      // 第一个字母在字母表中的序号，处理冲突的方法是线性探测开放定址。
      for (i=1; i<=26; i++) {
            j=i;
            while (ht.elem[j].key) {
                  if (Hash (ht.elem[j].key) == i) printf (ht.elem[j].key);
                  j= (j+1)% m;
            }
      }
} //PrintWord
```

33．假设哈希表长为*m*，哈希函数为*H*(*x*)，用链地址法处理冲突。试编写输入一组关键字并建造哈希表的算法。

【解答】算法如下：

```
typedef *LNode[MAXSIZE] CHashTable; //链地址Hash表类型
Status Build_Hash(CHashTable &T，int m)//输入一组关键字，建立Hash表，表长为m，用链地址法处
                                        //理冲突。
{
  if(m<1) return ERROR;
  T=malloc(m*sizeof(WORD)); //建立表头指针向量
  for(i=0;i<m;i++) T[i]=NULL;
  while((key=Inputkey())!=NULL) //假定Inputkey函数用于从键盘输入关键字
  {
    q=(LNode*)malloc(sizeof(LNode));
    q->data=key;q->next=NULL;
    n=H(key);
    if(!T[n]) T[n]=q; //作为链表的第一个结点
    else
    {
      for(p=T[n];p->next;p=p->next);
      p->next=q; //插入链表尾部，本算法不考虑排序问题
    }
  }//while
  return OK;
}//Build_Hash
```

34．假设有一个1000×1000的稀疏矩阵，其中1%的元素为非零元素，现要求用哈希表作存储结构。试设计一个哈希表并编写相应算法，对给定的行值和列值确定矩阵元素在哈

希表上的位置。请将你的算法与在稀疏矩阵的三元组表存储结构上存取元素的算法（不必写出）进行时间复杂度的比较。

【解答】算法如下：

```
Status Locate_Hash(HashTable H，int row，int col，KeyType key，int &k)
            //根据行列值在Hash表表示的稀疏矩阵中确定元素key的位置k
{
    h=2*(100*(row/10)+col/10); //设计的Hash函数
    while(H.elem[h].key&&!EQ(H.elem[h].key，key))
        h=(h+1)%20000;
    if(EQ(H.elem[h].key，key)) k=h;
    else k=NULL;
}//Locate_Hash
```

本算法所使用的Hash表长20000，装填因子为50%，Hash函数为行数前两位和列数前两位所组成的四位数再乘以二，用线性探测法处理冲突。当矩阵的元素是随机分布时，查找的时间复杂度为$O(1)$。

9.4 考研真题分析

例题9-1 （中国科学院计算技术研究所1999年试题）

最优二叉树（哈夫曼树），最优查找树均为平均查找路径长度最小的树，其中对最优二叉树，n表示__A__；对最优查找树，n表示__B__，构造这两种树均__C__。

（1）结点数 （2）叶子结点数 （3）非叶子结点数 （4）度为2的结点数 （5）需要一张n个关键字的有序表 （6）需要对n个关键字进行动态插入 （7）需要n个关键字的查找概率表 （8）不需要任何前提

【分析】最优二叉树（哈夫曼树）是对叶子结点带权平均查找路径长度最小的树，最优查找树是对所有结点带权查找路径长度最小的树，构造这两种树均需要知道关键字的查找概率。

【解答】A.（2） B.（1） C.（7）

例题9-2 （中国科学院计算技术研究所1999年试题）

m路B+树是一棵__A__，其结点中关键字最多为__B__个，最少为__C__个。

（1）m路平衡查找树 （2）m路平衡索引树 （3）m路trie树 （4）m路键树

（5）$m-1$ （6）m （7）$m+1$ （8）$\left\lceil \frac{m}{2} \right\rceil -1$ （9）$\left\lceil \frac{m}{2} \right\rceil$ （10）$\left\lceil \frac{m}{2} \right\rceil +1$

【解答】A.（2） B.（6） C.（9）

【扩展】注意B+树与B-树的区别。

B-树是一种平衡的多路查找树，一棵m阶的B-树，或为空树，或为满足下列特征的m叉树；

① 树中每个结点至多有m棵子树，

② 若根结点不是叶子结点，则至少有两棵子树，

③ 除根之外的所有非终端结点至少有$\left\lceil \frac{m}{2} \right\rceil$棵子树，

④ 所有的非终端结点中包含下列信息数据：

（n，A_0，K_1，A_1，K_2，A_2，…，K_n，A_n）

其中，K_i(i=1，…，n)为关键字，且K_i<K_i+1，A_i为指向子树根结点的指针，且指针A_{i-1}所指子树中所有结点的关键字均小于K_i，A_n所指子树中所有结点的关键字均大于K_n，n为关键字的个数。

⑤ 所有的叶子结点都出现在同一层次上，并且不带信息。

B+树是应文件系统所需而出的一种B-树的变型树，一棵m阶的B+树和m阶的B-树的差异在于：

① 有n棵子树的结点中含有n个关键字。

② 所有的叶子结点中包含了全部关键字的信息及指向含这些关键字的记录的指针且叶子结点本身按关键字的大小自小而大顺序连接。

③ 所有的非终端结点可以看成是索引部分，结点中只含有其子树（根结点）中的最大（或最小）关键字。

例题9-3 （中国科学院计算技术研究所1999年试题）

具有n个关键字的B-树的查找路径长度不会大于_____。

【分析】在B-树中，除根结点外所有非终端结点至少含有$\left\lceil \frac{m}{2} \right\rceil$棵子树，故含有$n$个关键字的B-树的最大深度为根结点具有两棵子树，其余的非终端结点都含有$\left\lceil \frac{m}{2} \right\rceil$棵子树，此时其深度为$1+\log_{\left\lceil \frac{m}{2} \right\rceil}\left(\frac{n+1}{2}\right)$。

【解答】$1+\log_{\left\lceil \frac{m}{2} \right\rceil}\left(\frac{n+1}{2}\right)$

例题9-4 （中国科学院软件研究所1999年试题）

二叉排序树采用二叉链表存储。写一个算法，删除结点值是X的结点。要求删除该结点后，此树仍然是一棵二叉排序树，并且高度没有增长。（注：可不考虑被删除的结点是根的情况。）

【解答】算法如下：

```
{查找值是X的结点}
FUNCTION bstsearch(VAR t: bstptr:bstbtr; X:keytype):Boolean;
```

```
BEGIN
    f:=t;
    WHILE t<>NIL DO
        DO CASE
            CASE t^.key=X
                return(true);
            CASE t^key>X
                f:=t;
                t:=t^.lchild
            CASE t^.key<X
                f:=t;
                t:=t^.rchild;
            ENDCASE
    ENDWHILE;
    return(false);
END;

{删除值是X的结点}
PROCEDURE delnode(VAR t:bstptr; X:keytype)
BEGIN
    p，  f: bstptr;
    {p为叶子结点}

        IF bstserach(p，f，X)=true THEN
            IF p=f THEN
            return;
        ENDIF
        IF(p^.lchild=NIL)AND (p^.rchild=NIL)THEN
            IF f^.lchild=p THEN f^.lchild:=NIL;
            ELSE f^.rchild:=NIL;
            ENDIF
            dispose(p);
            return;
        ENDIF

    {P没有左子树的情况}
        IF p^.lchild=NIL THEN
            IF f^.lchild=p THEN f^.lchild:=p^.rchild
            ELSE f^.rchild:=p^.rchild;
            ENDIF
            dispose(p);
            return;
        ENDIF

    {p没有右子树的情况}
        IF P^.rchild=NIL THEN
            IF f^.lchild=p THEN f^.lchild:=p^.lchild;
            ELSE f^.rchild:=p^.lchild;
            ENDIF
            dispose(p);
```

```
            return;
        ENDIF

    {p既没有左子树也没有右子树的情况}
        q:=p;
        s:=p^.lchild;

    {查找p的中序直接前驱s}
        WHILE s^.rchild<> NIL DO
            q:=s;
            s:=s^.rchild;
        ENDWHILE

    {用s的值代替p的结点值}
        p^.data:=s^.data;

    {删除s}
        IF q<>p THEN q^.rchild:=s^.lchild;
        ELSE q^.lchild:=s^.lchild;
        ENDIF
    dispose(s);
    return;

    ELSE
    error(“没有找到该结点”);
    ENDIF

END
```

例题9-5 （清华大学1998年试题）

使用哈希函数hashf(x)=x MOD 11，把一个整数值转换成哈希表下标，现要把数据1，13，12，34，38，33，27，22插入到哈希表中。

① 使用线性探测再散列法来构造哈希表。

② 使用链地址法来构造哈希表。

③ 针对这种情况，确定其装填因子，查找成功所需的平均查找次数以及查找不成功所需的平均探测次数。

【分析】主要考察用线性探测再散列法和链地址法构造哈希表。

哈希表的装填因子定义为：

α=表中添入的记录数/哈希表的长度

线性探测再散列的哈希表查找成功时的平均查找长度为：

$$S_{\mathrm{nl}} \approx \frac{1}{2}\left(1+\frac{1}{1-\alpha}\right)$$

线性探测再散列的哈希表查找不成功时的平均查找长度为：

$$U_{\mathrm{nl}} \approx \frac{1}{2}\left(1+\frac{1}{(1-\alpha)^2}\right)$$

使用链地址法查找成功的平均查找长度为：

$$S_{nc} \approx 1+\frac{\alpha}{2}$$

使用链地址法查找不成功所需的平均查找次数为：

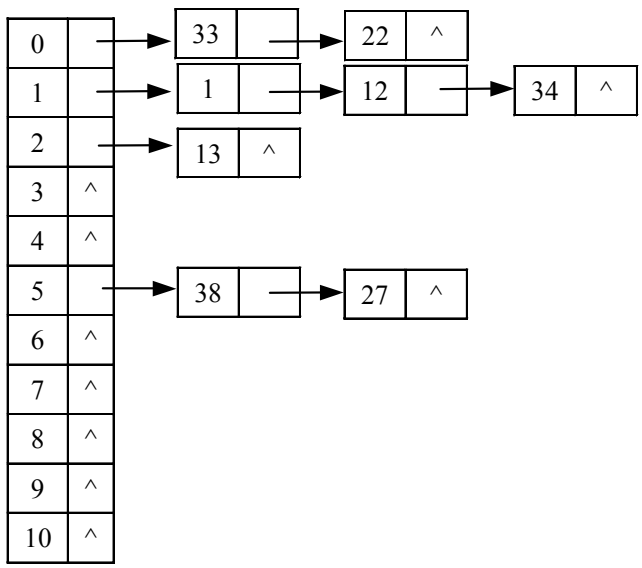
$$U_{nc} = \alpha + e^{-\alpha}$$

【解答】① 使用线性探测再散列法来构造哈希表见下表。

哈希表

地址	0	1	2	3	4	5	6	7	8	9	10
数据	33	1	13	12	34	38	27	22			

② 使用链地址法来构造哈希表如下图所示。



③ 装填因子α=8/11

使用线性探测再散列法查找成功所需的平均查找次数为：

$$S_{nl} \approx \frac{1}{2}(1+\frac{1}{1-\alpha}) = \frac{1}{2}(1+\frac{1}{1-\frac{8}{11}}) = \frac{7}{3}$$

使用线性探测再散列法查找不成功所需的平均查找次数为：

$$U_{nl} \approx \frac{1}{2}(1+\frac{1}{(1-\alpha)^2}) = \frac{1}{2}(1+\frac{1}{(1-\frac{8}{11})^2}) = \frac{65}{9}$$

使用链地址法查找成功所需的平均查找次数为：

$$S_{nc} \approx 1+\frac{\alpha}{2}=1+\frac{\frac{8}{11}}{2}=\frac{15}{11}$$

使用链地址法查找不成功所需的平均查找次数为：

$$U_{nc} \approx \alpha + e^{-\alpha} = \frac{8}{11} + e^{-\frac{8}{11}}$$

【扩展】随机探测再散列、二次探测再散列和再哈希表查找成功时的平均查找长度为：

$$S_{nr} \approx -\frac{1}{\alpha}\ln(1-\alpha)$$

随机探测再散列、二次探测再散列和再哈希的哈希表查找不成功时的平均查找长度为：

$$U_{nr} \approx \frac{1}{1-\alpha}$$

例题9-6 （中国科学院软件研究所1999年试题）

判断正误：
负载因子（装填因子）是哈希表的一个重要参数，它反映哈希表的装满程度。

【解答】正确。

例题9-7 （中国科学院软件研究所1998年试题）

假设客观存在有K个关键字互为同义词，若用线性探测法把这K个关键字存入哈希表中，至少要进行_____次探测。

A. K-1　　B. K　　C. K+1　　D. $K(K+1)/2$

【分析】至少需进行1+2+…+K=K(K+1)/2次探测。
【解答】D

例题9-8 （清华大学1998年试题）

给定整型数组B[0..m，0..n]。已知B中数据在每一维方向上都按从小到大的次序排列。且整型变量x在B中存在。试设计一个程序段，找出一对满足B[i，j]=x的i，j，要求比较次数不超过m+n。

【分析】本题中主要是要确定每次进行比较的对象。其中二维数组右上角的元素是一个较为特殊的元素。可以逐次跟二维数组右上角的元素进行比较。每次比较有3种可能的结果：若相等则结束比赛；若右上角的元素小于x，则可断定二维数组的最右面一列中肯定不包含x，下次比较时搜索范围即可减少一列。这样，每次至少可使搜索范围减少一列或一行，最多经过m+n次即可找到x。

【解答】程序如下：

```
PROC find(B:ARRAY[0..m，0..n] OF integer; x: integer; VAR i，j:integer);
{返回i，j 使得B[i，j]=x}
        i:=0; j:=n;
```

```
        WHILE (B[i，j]<>x) DO
              IF (B[i，j]<x)
              i:=i+1;
              ELSE IF(B[i，j]>x)
              j:=j−1;
    ENDP; {find}
```

例题9−9 （东北大学1998年试题）

已知记录关键字集合为（53，17，19，61，98，75，79，63，46，49），要求散列到地址区间（100，101，102，103，104，105，106，107，108，109）内，若产生冲突用开型寻址法的线性探测法解决，要求写出选用的哈希函数；形成的哈希表；计算出查找成功时平均查找长度与查找不成功的平均查找长度（设等概率情况）。

【解答】哈希函数：$H(key)$=100+(*key*个位数+*key*+位数) mod 10;

形成的哈希表：

100	101	102	103	104	105	106	107	108	109
19	98	75	63	46	49	79	61	53	17

查找成功时的平均长度为(1+2+1+1+5+1+1+5+5+3)/10=2.5次

查找不成功时的平均长度为(1+4+4+4+2)/10=1.5次

例题9−10（清华大学1999年试题）

设有12个数据（25，40，33，47，12，66，72，87，94，22，5，58），它们存储在哈希表中，利用再哈希解决冲突，要求插入新数据的平均查找次数不超过3次。

① 该哈希表的大小*m*应设计为多大?

② 试为该哈希表设计相应的哈希函数（用除留余数法）和计算寻找下一个“空位”时向前跨步步长的再散列函数。

③ 顺次将各个数据散列到表中。

④ 计算查找成功的平均查找次数。

【解答】

① 线性探测再散列的哈希表查找成功的平均查找长度为

$$S_{\mathrm{nl}}=\frac{1}{2}\left(1+\frac{1}{1-\alpha}\right)\leq 3$$

得α≤4/5 （α为填充因子）

即12/*m*≤4/5，所以*m*≥15

不妨取*m*=16

② 哈希函数$H(key)$=*key* mod 16

再散列函数为$H(key)$+3。

③ 形成的哈希表为

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	33	66			5	22	87	40	25	58	72	12		94	47

④ 查找成功的平均查找次数为：

（1+1+1+…+1+2+1）/12=1.08

例题9-11 （北京邮电大学1998年试题）

有关键字集合K={15，22，50，13，20，36，28，48，35，31，41，18}采用散列存取，哈希表为HT[0..14]。设哈希函数$H(K)=K$ MOD 13，解决冲突采用开放定址法中的二次探测再散列的方法。试将K值填入HT表中，并把查找每个关键字所需比较次数m填入下表，并请计算出查找成功时的平均查找长度。

HT表

1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
K															
m															

【解答】将查找每个关键字所需比较次数m填入表中，如下所示。

填入比较次数后的HT值

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
K	13		15	28		18	41	20	48	22	36	50		35	
m	1		1	2		1	4	1	3	1	1	1		4	

查找成功时的平均查找长度为：

（1+1+2+1+4+1+3+1+1+1+4）/12=$\frac{5}{3}$

例题9-12（清华大学1999年试题）

设有5个数据do、for、if、repeat和while，它们排在一个有序表中，其查找概率分别为p_1=0.2，p_2=0.15，p_3=0.14，p_4=0.03，p_5=0.01，而查找它们之间不存在数据的概率分别为q_0=0.2，q_1=0.15，q_2=0.1，q_3=0.03，q_4=0.02，q_5=0.01。

do for if repeat while

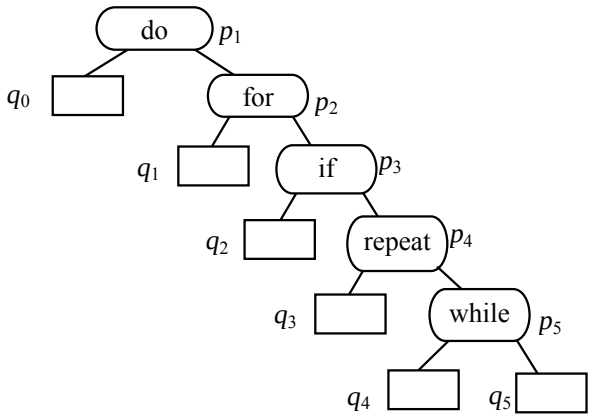
q_0 p_1 q_1 p_2 q_2 p_3 q_3 p_4 q_4 p_5 q_5

① 试画出对该有序表采用顺序查找时的判定树和采用折半查找时的判定树。

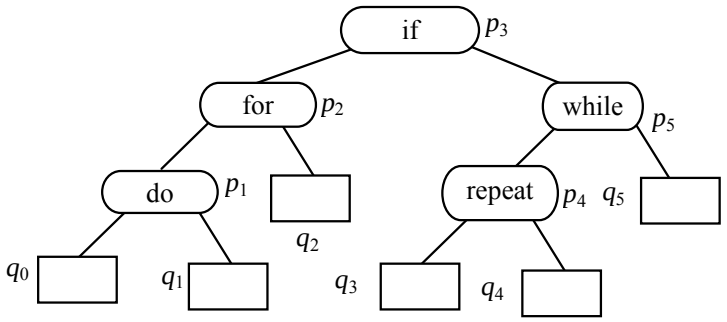
② 分别计算顺序查找时的查找成功和不成功的平均查找长度以及折半查找时的查找成功和不成功的平均查找长度。

③ 判定是顺序查找好，还是折半查找好。

【解答】① 顺序查找时的判定树如下图所示。



折半查找时的判定树如下图所示。



② 顺序查找

成功平均查找长度为

$(1*p_1+2*p_2+3*p_3+4*p_4+5*p_5)/(p_1+p_2+p_3+p_4+p_5)=2.057$

不成功平均查找长度为

$(1*q_0+2*q_1+3*q_2+4*q_3+5*q_4+5*q_5)/(q_0+q_1+q_2+q_3+q_4+q_5)=2.098$

折半查找

成功平均查找长度为

$(1*p_3+2*p_2+2*p_5+3*p_1+3*p_4)/(p_1+p_2+p_3+p_4+p_5)=2.170$

不成功平均查找长度为

$(3*q_0+3*q_1+2*q_2+3*q_3+3*q_4+2*q_5)/(q_0+q_1+q_2+q_3+q_4+q_5)=2.784$

③ 顺序查找好。

例题9-13 （清华大学1999年试题）

在分析二叉查找树性能时常加入失败结点，即外结点，从而形成扩充的二叉树。若设失败结点i所在层次为l_i，那么查找失败到达失败结点时所做的数据比较次数是多少？

【解答】（i-1）次。

例题9-14（北京邮电大学1998年试题）

判断正误：

有n个数存放在一维数组A[1..n]中，在进行顺序查找时，这n个数的排序有序或无序其平均查找长度不同。

【解答】错误。

顺序查找并没有假设数是有序或者无序，因此有序或无序对平均查找长度没有影响。

例题9-15　（清华大学2002年试题）

判断正误：

（1）3阶的B-树是平衡的3路搜索树。反之，一棵平衡的3路搜索树是3阶B-树。

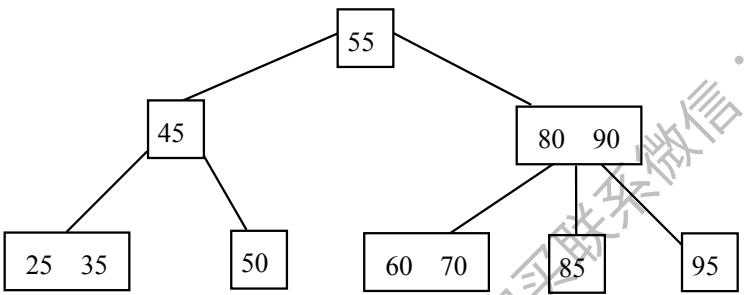
（2）随着装填因子α的增大，用闭散列法解决冲突，其平均搜索长度比用开散列法解决冲突时的平均搜索长度增长得慢。

【解答】（1）错误　　（2）正确

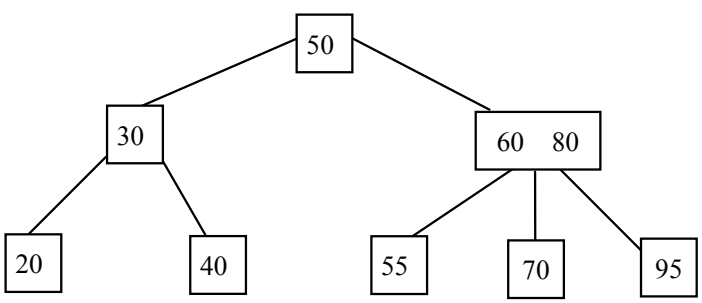
例题9-16　（清华大学2002年试题）

关于B-树的简作题

（1）下图是一个3阶B-树。试画出插入65、15、40、30后B-树的变化。

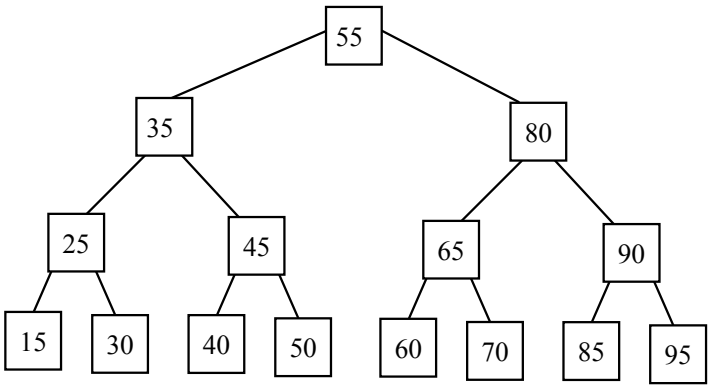


（2）下图是一个3阶B-树。试分别画出在删除50、40之后B-树的变化。

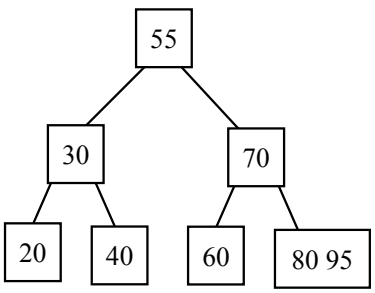


【解答】

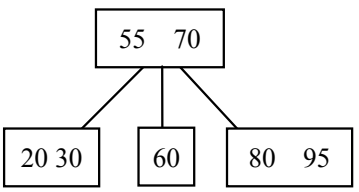
（1）插入65、15、40、30后B-树变为如下所示：



（2）删除50后B-树为



删除40后B-树为：



例题9-17 （中国科学院计算机网络信息中心2001年试题）

单选题：

哈希表的平均查找长度和_____无直接关系。

A．哈希表记录类型　　B．哈希函数

C．处理冲突的方法　　D．装填因子

【解答】A

例题9-18 （中国科学院计算机网络信息中心2001年试题）

问答题：

对于给定的关键字序列，若哈希函数无冲突，则称其为完备的。设哈希表长度为7，试为（Btet，Jane，Shirley，Bryce，Michelle，Heather）设计一个完备的哈希函数H（提示：考虑每个字符串的第3个字符），并写出其C代码。

【解答】

```
int H(char *S) {
        return (S[2]-'a')%7;
}
```

例题9-19 （中国科学院计算机网络信息中心2000年试题）

在一棵含有n个关键字的m阶B-树中进行查找，至多读盘_____次。

A．$\log_2 n$　　B．$1+\log_2 n$　　C．$1+\log_{\lceil m/2 \rceil}\frac{n+1}{2}$　　D．$1+\log_{\lceil n/2 \rceil}\frac{m+1}{2}$

【解答】C

例题9-20 （华北计算技术研究所2001年试题）

简述二叉排序树的性质，并用序列（46，88，45，39，70，58，101，10，66，34）建立一个二叉排序树，画出该树，求在等概率情况下，查找成功的平均查找长度。

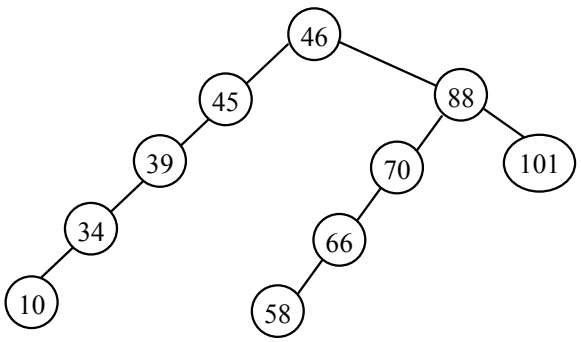
【解答】二叉排序树的性质为：

（1）若其左子树不为空，则左子树上所有结点的值均小于其根结点；

（2）若其右子树不为空，则右子树上所有结点的值均大于其根结点；

（3）其左、右子树分别为二叉排序树。

该二叉排序树如图所示：



平均查找长度=$\frac{1+2\times2+3\times3+4\times2+5\times2}{10}=3.2$

例题9-21 （华北计算技术研究所2001年试题）

下列算法在平衡二叉树上插入一个结点，读该算法并回答：

（1）简述平衡二叉树的性质，解释平衡因子的作用。

（2）给出算法中LR-rotation所表示的LR型旋转变换时所作的一组指针修改语句。

（3）给出算法中RR-rotation所表示的RR型旋转变换时所作的一组指针修改语句。

（4）在[　　]中填上合适的语句，包括注释语句。

程序如下：

```
PROC ins_AVLtree(VAR t:AVL inktp;K:keytp);
{在以结点t为根的平衡树上插入关键字等于K的新结点s}
    new(s); s↑.key:=K;s↑.lchild:=s↑.rchild:=NIL;s↑.bf:=0;
    IF t=NIL THEN t:=s;          {插入根结点}
    ELSE  [{查找s↑的插入位置，并记下a}
        f:=NTL;a:=t;p:=t;q:=NIL;
        WHILE p≠NIL DO
            [IF p↑.bf≠0 THEN [a:=p;f:=q];
            q:=p;
            IF s↑.key<p↑.key THEN p:=p↑.lchild
            ELSE p↑:=p.rchild;
        ];{      ①      }
```

```
        IF s↑.key<q↑.key THEN q↑.lchild:=s;
        ELSE q↑.rchild:=s;{插入s↑}
        {修改a↑至s↑路径上各结点的平衡因子值}
        IF   s↑.key< a↑.key THEN
             [      ②      ;b:=p;d:=1]{ s↑插入在a↑的左子树中。}
        ELSE [p:=p↑.rchild;b:=p;    ③    ] ;
        WHILE p≠s   DO
             IF s↑ .key<p↑.key THEN
                  [p↑.bf:=1;p:=p↑.lchild] {左子树深度增1}
             ELSE [    ④    ; p:=p↑.rchild];
        {判别以a为根的子树是否失去平衡}
        balanced:=true;
        CASE
             a↑.bf=0:          a↑.bf:=d;
             a↑.bf+d=0:        a↑.bf:=0;
             ELSE{失去平衡，判别旋转类型}
                  [balanced:=false;
                   IF d=+1   THEN
             CASE
                       b↑.bf=1:     LL-rotation;
                       b↑.bf=-1:   LR-rotation
                  ENDC
             ELSE
                  CASE
                       b↑.bf=-1:     RR-rotation;
                       b↑.bf=1:      RL-rotation
                  ENDC
                 ]
        ENDC;
        IF NOT balanced THEN
                  CASE{在RL和LR旋转处理结束时令b:=c}
                  f=NIL;                  t:=b;
                  f↑.lchild=a:   ⑤   ;
                  f↑.rchild=a: f↑.rchild:=b
        ENDC;   {修改a↑的双亲f↑的指针域}

    ]
ENDP; {ins_AVLtree}
```

【解答】

（1）平衡二叉树或者是一棵空树，或者是它的左子树和右子树都是平衡二叉树，且左子树和右子树的深度之差的绝对值不超过1。平衡因子是对二叉树上的结点而言，定义该结点的左子树深度减去右子树的深度，在平衡二叉树上，其平衡因子的值只可能为-1、0、1，若其绝对值大于等于2，则该二叉树不平衡。

（2）LR-rotation变换

```
b:=a↑.lchild;   c:=b↑.rchild;
a↑.lchild:=c↑.rchild;   b↑.rchild:=c↑.lchild;
c↑.rchild:=a;    c↑.lchild:=b;{c为子树新根}
```

```
case{插入之前c.bf为0，插入之后有三种情况}
      c↑.bf=1:  [a↑.bf:=-1;   b↑.bf:=0]
      c↑.bf=-1: [a↑.bf:=0;    b↑.bf:=1]
      c↑.bf=0:  [a↑.bf;=0;    b↑.bf:=0]
ENDC;
c↑.bf:=0;          b:=c;{旋转变化之后}
```

（3）RR-ratation变换

```
b:=a↑.rchild;
a↑.rchild:=b↑.lchild;    a↑.bf:=0;
b↑.lchild:=a;      b↑.bf:=0;
```

（4）填空如下：

① 找到插入位置q↑

② p:=a↑.lchild

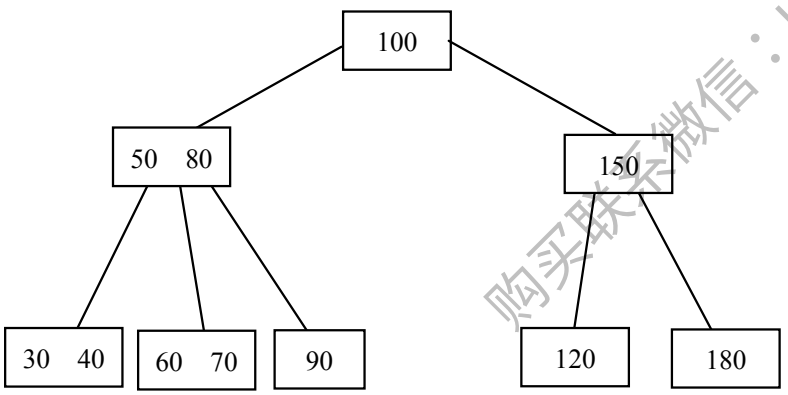
③ d:=-1

④ p↑.bf:=-1

⑤ f↑.lchild:=b

例题9-22 （北京大学2000年试题）

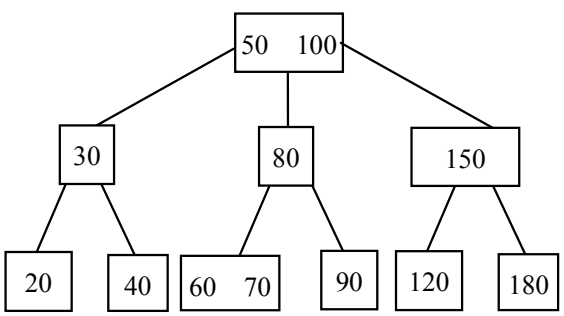
设有3阶B树如下图所示：



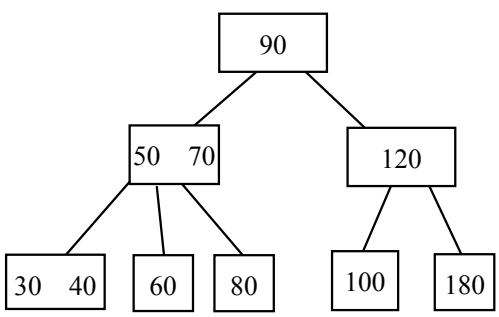
分别画出从上图所示B树中插入关键字20后和从上图所示B树中删除关键字150后得到的两个B树。

【解答】

插入20后



删除150后



例题9-23 （中国人民大学2002年试题）

以下是折半插入排序算法，请在留空处填入适当内容：

```
void binsort(int k[],  int n)
  {int i，j，b，e，m，key;  /*对数组k中的n个结点从小到大排序*/
    for (i=1;i<n;i++)
       {j=i-1;
         if(  ①  )
            {b=0;e=j;key=k[i];
             while(  ②  )
               {m=(b+e)/2;
                 if(key<k[m])  ③ ;
                 else if(key>k[m])  ④ ;
                       else{  ⑤  }
               }
             while(  ⑥  )
                {k[j+1]=k[j];j=j-1;}
             k[  ⑦  ]=key;
             }
       }
  }
```

【解答】

① j>0　② e>=b　③ e=m-1　④ b=m+1　⑤ break;　⑥ j>=m　⑦ m

例题9-24 （北京科技大学2002年试题）

在含有$n(n\geqslant 0)$个关键字的m阶B-树上查找时，查找路径上最多涉及多少个结点？

【解答】$\log_{\lceil m/2\rceil}(\frac{N+1}{2})+1$

例题9-25 （武汉理工大学2002年试题）

选择题：

二叉排序树是_____。

供选择的答案：

A．中序遍历得到一升序序列的二叉树

B．每一分支结点的度均为2的二叉树

C．按层次从左到右顺序编号的二叉树

D．每一分支结点的值均小于左子树上所有结点的值（若左子树存在），又大于右子树上所有结点的值（若右子树存在）。

【解答】A

例题9-26 （武汉理工大学2002年试题）

判断题：

装填（负载）因子是哈希法的一个重要参数，它反映哈希表的装满程度。

【解答】正确。

例题9-27 （武汉理工大学2002年试题）

判断题：

在二叉查找树中插入的新结点，总是被插入到某个叶子结点的下面。

【解答】错误。

例题9-28 （武汉理工大学2002年试题）

简答题：

在地址空间为0~16的散列区中，对以下关键字序列

（Jan，Feb，Mar，Apr，May，Jun，Jul，Aug，Sep，Oct，Nov，Dec）

（1）构造哈希表。$H(x)=i \bmod 17$ (或$H(x)=i \% 17$)，其中i为关键字中第一个字母在字母表中的序号，解决冲突的方法用线性探测法。

（2）求出查找成功和查找失败的平均查找长度。

【解答】（1）哈希表如下所示

地址	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
数据	Apr	Aug	Sep		Dec		Feb				Jan	Jun	Jul	Mar	May	Oct	Nov

（2）哈希表的装填因子 $\alpha=\dfrac{12}{17}$

查找成功的平均查找长度 $S_{\mathrm{nl}}=\dfrac{1}{2}\left(1+\dfrac{1}{1-\alpha}\right)=\dfrac{11}{5}$

查找失败的平均查找长度 $U_{\mathrm{nl}}=\dfrac{1}{2}\left(1+\dfrac{1}{(1-\alpha)^2}\right)=\dfrac{157}{25}$

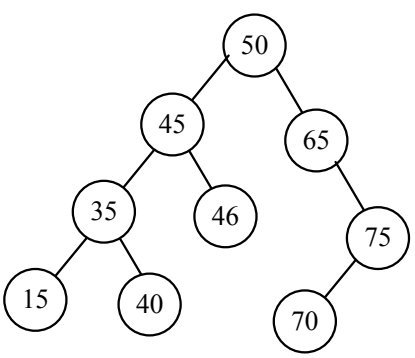
例题9-29 （南京理工大学2002年试题）

简答题：

构造一棵二叉排序树，对其进行前序遍历可得到如下元素序列：

50，45，35，15，40，46，65，75，70

【解答】二叉排序树为：



错误！

例题9-30 （南京理工大学2002年试题）

选择题：

设$H(x)$是一哈希函数，有K个不同的关键字（x_1，x_2，…，x_k）满足$H(x_1)=H(x_2)=\cdots=H(x_k)$，若用线性探测法将这$K$个关键字存入哈希表中，至少要探测_____次。

A．$K-1$　　B．K　　C．$K+1$　　D．$K(K-1)/2$

【解答】D

例题9-31 （南京理工大学2002年试题）

选择题：

在一棵m阶B树中，若在某结点中插入一个新关键字而引起该结点分裂，则此结点中原有__（1）__个关键字；若在某结点中删去一个关键字而导致结点合并，则该结点中原有的关键字的个数是__（2）__。

（1）A．m　　B．$\lceil m/2\rceil-1$　　C．$m+1$　　D．$m-1$

（2）A．$m-1$　　B．$\lceil m/2\rceil-1$　　C．$m+1$　　D．m

【解答】（1）D　　（2）B

例题9-32 （南京理工大学2002年试题）

选择题：

具有n个数据元素的顺序组织的表，一个递增有序，一个无序，查找一个元素时采用顺序算法，对有序表从头开始查找，发现当前检测元素已不小于待查元素时，停止检索，确定查找不成功。已知查找任一元素的概率是相同的，则在两种表中成功查找_____。

A．平均时间后者小　　B．无法确定

C．平均时间两者相同　　D．平均时间前者小

【解答】C

9.5 自 测 题

自测题9-1

设有一组数据black，blue，green，purple，red，white，yellow，它们的查找概率分别是0.10，0.08，0.12，0.05，0.20，0.25，0.20，试以它们的查找概率为权值，构造一棵次查找树，并计算其查找成功的平均查找长度。

自测题9-2

判断正误：

向量和单链表表示的有序表均可使用折半查找方法来提高查找速度。

自测题9-3

在关键字随机分布的情况下，用二叉排序树的方法进行查找，其查找长度与_____量级相当。

A. 顺序查找　　B. 折半查找　　C. 前两者均不正确

自测题9-4

在非空m阶B-树上，除根结点以外的所有其他非终端结点_____。

A. 至少有$\left\lfloor \frac{m}{2} \right\rfloor$棵子树　　B. 至多有$\left\lfloor \frac{m}{2} \right\rfloor$棵子树

C. 至少有$\left\lceil \frac{m}{2} \right\rceil$棵子树　　D. 至少有$\left\lceil \frac{m}{2} \right\rceil$棵子树

自测题9-5

一棵深度为k的平衡二叉树，其每个非终端结点的平衡因子均为0，则该树共有_____个结点。

A. $2^{k-1}-1$　　B. 2^{k-1}　　C. $2^{k-1}+1$　　D. 2^k-1　　E. 2^k　　F. 2^k+1

自测题9-6

已知序列17，31，13，11，20，35，25，8，4，24，40，27，请画出该序列的二叉排序树，并分别给出下列操作后的二叉树；

① 插入数据9；

② 删除结点17；

③ 再删除结点13。

自测题9-7

判断正误：

在任意一棵非空二叉排序树中，删除某结点后又将其插入，则所得二叉排序树与删除前原二叉排序树相同。

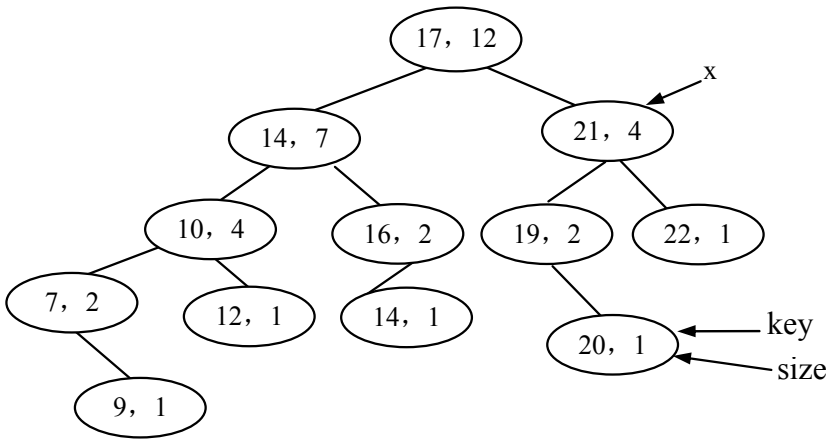
自测题9-8

设二叉排序树的存储结构为：

```
TYPE tree=^node;
node=RECORD;
   key:keytype;
size:int;
lchild，rchild，parents:tree;
END;
```

一个结点x^的size域的值是以该结点为根的子树中结点的总数（包括x^本身）。例如，图9.7中x所指结点的size值为4。设树高为h，试写一时间复杂度为$O(h)$的算法rank(T:tree;x:^node) 返回x所指结点在二叉排序树T的中序序列里的排序序号，即：求x^结点是根为T的二叉排序树中第几个最小元素。例如，下图中x所指结点是树T中第11个最小

元素。（提示：可利用size值和双亲指针parents。）



自测题9-9

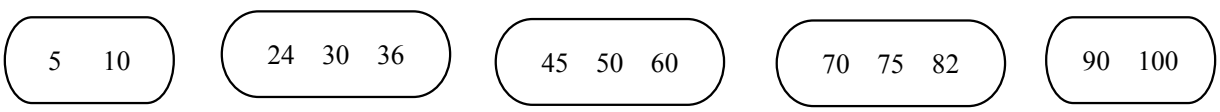
何谓Trie树？试构造一棵对应于下列关键字集的Trie树，请注意应使树的深度尽可能小。{program，programmer，programming，processor，or}

自测题9-10

若哈希（Hash）表的地址范围为[0，9]，哈希函数为$H(key)=(key^2+2)$MOD 9，并采用链地址法处理冲突，请画出元素7、4、5、3、6、2、8、9依次插入哈希表以后该哈希表的状态。

自测题9-11

已知一个B+树有5个叶子结点，每个叶子结点中的关键字如下：



请先画出这棵3阶B+树，然后在你给出的3阶B+树中插入关键字65，再画出插入后的B+树。

自测题9-12

在分块检索中，对256个元素的线性表分成_____块最好，每块的最佳长度是____；若每块的长度为8，其平均检索长度为_____。

自测题9-13

已知二叉排序树采用二叉链表存储结构，根结点的指针为T，链结点的构造为：

lchild	data	rchild

其中lchild，rchild分别指向该结点左、右孩子的指针（当孩子结点不存在时，相应指针域为nil），data域存放结点的数据信息。请写出递归算法，从小到大输出该二叉排序树中所有数据值大于等于x的结点的数据。要求先找到第一个满足条件的结点后再依次输出其他满足

条件的结点。

自测题9-14

判断正误：

① 最佳二叉树是AVL树（平衡二叉树）。

② 二叉树中，具有两个子女的结点的中序后继结点最多只能有一个子女。

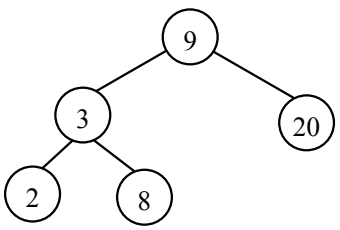
③ 若哈希表的负载因子α<1，则可避免碰撞的产生。

自测题9-15

设有关键码A、B、C和D，按照不同的输入顺序，共可能组成多少不同的二叉排序树。请画出其中高度较小的6种。

自测题9-16

在如下图所示的AVL树中，依次插入关键码为6和10的两个结点，请分别画出依次插入后的AVL树。



自测题9-17

编写对有序表进行顺序查找的算法，并画出对有序表进行顺序查找的判定树，假设每次查找时的给定值为随机值，又查找成功和不成功的概率也相等，试求进行每一次查找时，与给定值进行比较的关键字个数的期望值。

自测题9-18

含12个结点的平衡二叉树的最大深度是_____（设根结点深度为1），并画出一棵这样的树。

自测题9-19

解下面二叉排序树的有关问题：

① 有一组关键字为{7，5，8，16，12，18，14，6，4，10，9，13，15，17，20}，为使查找速度最快，需要构造最佳二叉排序树（平均查找长度最小），试画出此二叉排序树，并说明其基本思想（方法）。

② 优化下面的递归算法，减少进栈、出栈次数。

```
PROCEDURE bfind(p: bptr; K: keytype);
{在指针p所指的二叉排序树上查找其关键字等于给定值K的记录，当查找成功时返回所找结点的指针}
BEGIN
IF p!=NIL
    THEN IF p^.key=K
```

```
    THEN return(p){查找成功}
    ELSE IF K<P^.key
      THEN bfind(p^.Lc，K);
  ELSE bfind(p^.Rc，K)
ELSE return(p)   {查找失败，返回空指针}
END;
```

其中二叉排序树的结点结构为：

Lc	Key	Rc
左指针	关键字	右指针

自测题9-20

① 字符树中从根到一个树叶的路径上的所有结点连接起来构成（　　）。

② 在分块检索中，对256个元素的线性表分为（　　）块最好，每块的最佳长度为（　　）；若每块的长度为8，其平均检索长度为（　　）。

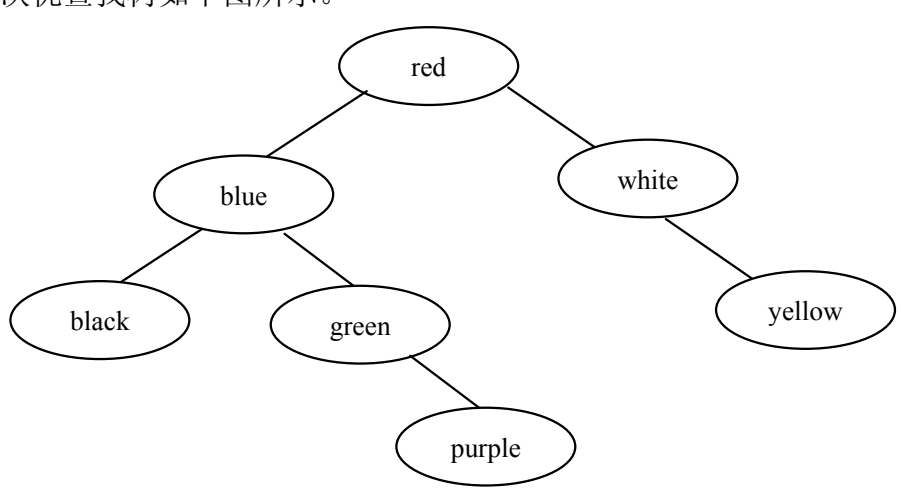
③ 静态索引结构是指在系统运行时（　　）不会发生改变。

9.6　自测题答案

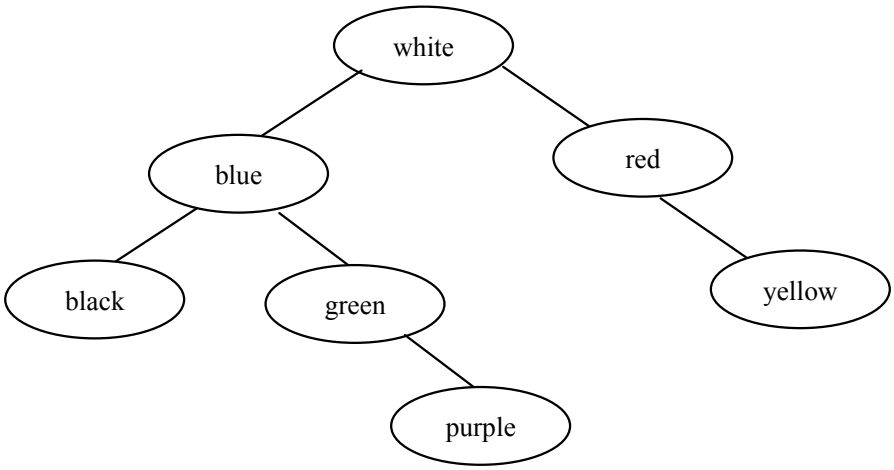
【自测题9-1】

j	1	2	3	4	5	6	7
sw_j	0.10	0.18	0.30	0.35	0.55	0.80	1.00
$\triangle P_j$	0.90	0.72	0.52	0.35	0.10	0.35	0.80
(根)					↑i		
$\triangle P_j$	0.25	0.07	0.13	0.30		0.20	0.25
(根)		↑i				↑i	
$\triangle P_j$			0.05	0.12			
(根)			↑i				

相应的次优查找树如下图所示。



进一步调整后的次优查找树如下图所示。



查找成功的平均查找长度=1×0.2＋2×(0.12+0.25)+3×(0.10+0.05+0.20)+4×0.08=2.31

【自测题9-2】

链表表示的有序表不能用折半查找法查找。

答案为错误。

【自测题9-3】

在随机的情况下，二叉排序树的平均查找长度为1+4log$n$，与折半查找同数量级。

答案为B。

【自测题9-4】

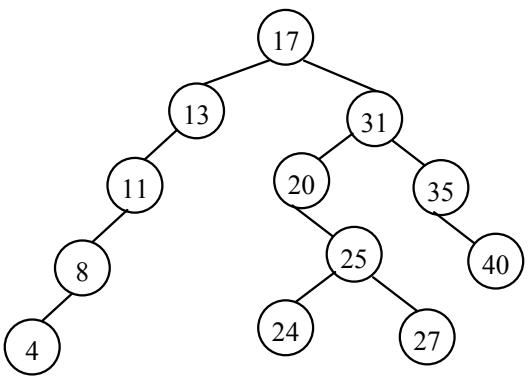
答案为C。

【自测题9-5】

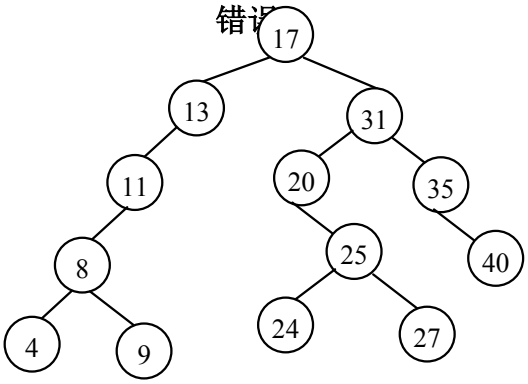
答案为D。

【自测题9-6】

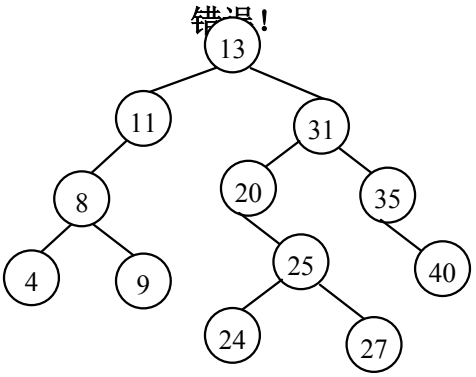
该序列的二叉排序树如下图所示。



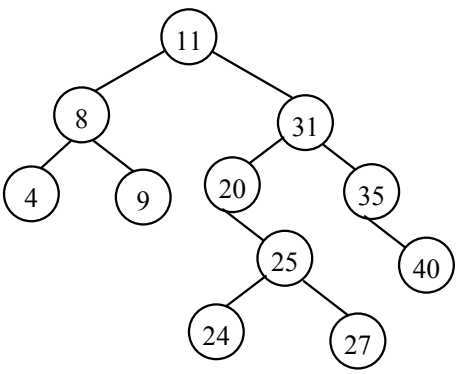
① 插入数据9后的二叉排序树如下图所示。



② 删除结点17后的二叉排序树如下图所示。



③ 再删除结点13后的二叉排序树如下图所示。



【自测题9-7】

除非被删除的结点是叶子结点，否则删除后再插入同一结点得到的二叉排序树与原来的二叉排序树不同。

答案为错误。

【自测题9-8】

此题要点：先计算x^在以x^为根的子树中的排序序号r，因为x^的左子树中所有结点在中根序列中均处于x^之前，所以有

r:=x^.左子树的大小（size）+1

这里加1是包括x^本身，当x^左子树为空时，相当于左子树size为0。

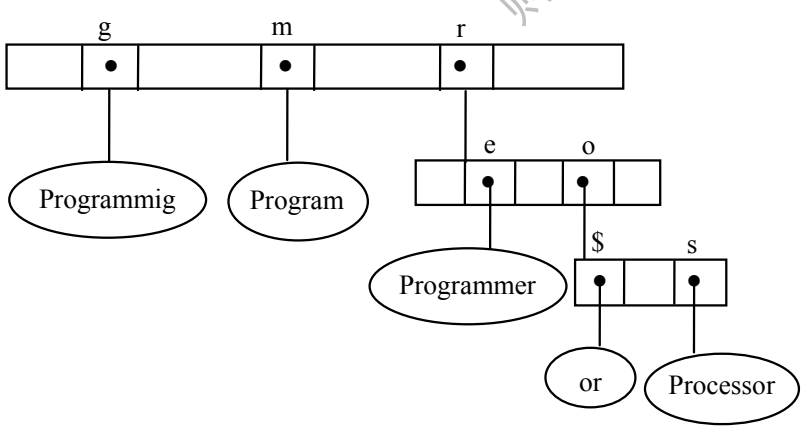
注意：x^在以x^上溯到其双亲，当x^是其双亲的左子树时，r值不变；当x^是其双亲右子树时，必须加上双亲结点本身及其左子树中的所有结点数作为新的r值。此过程最多上溯至根，所以执行时间不超过树的高度。

```
FUNC rank(T: tree ; x:^node):int
{
    //可设x≠NIL
    IF x^.lchild=NIL THEN
    r:=1;
    ELSE
        r:=x^.lchild^.size+1;
    y:=x;
    WHILE   y≠T DO{
        //循环不变量：r表示以y为根的子树中，结点x^的Rank值
        //当y等于根T时，即为所求
        IF   y:=y^.parents^.rchild THEN{
            //当y^是其双亲的右子树时，y^的双亲结点的左子树中所有结
            //点及y^的双亲本身均应排在y^的前面（指中根序列）
            IF y^.parents^.lchild=NIL THEN
                r:=r+1;
        }ELSE r:=r+y^.parents^.lchild^.size+1;
        y:=y^.parents;
    }//END OF child
    return r;
} //END OF FUNC
```

【自测题9-9】

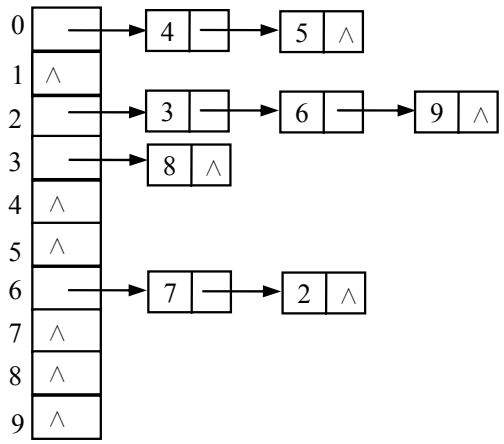
Trie树：以多重链表形式表示的键树，称为Trie树。

构造Trie树：对关键字集作如下分割，首先按尾字符不同分成若干子集，然后按倒数第二的字符不同分成若干子集，结果如下图所示。



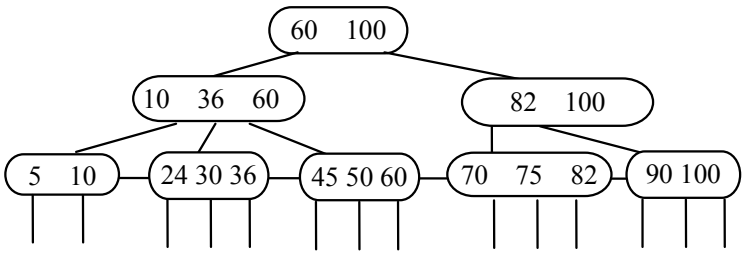
【自测题9-10】

将7、4、5、3、6、2、8、9依次插入哈希表以后，哈希表的状态如下图所示。

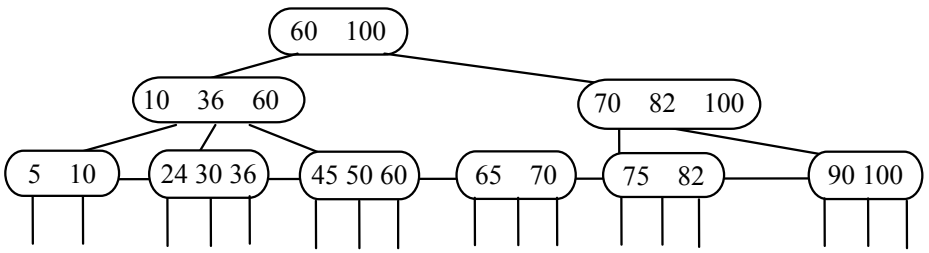


【自测题9-11】

该3阶B+树如下图所示。



插入65之后，B+树下图所示。



【自测题9-12】

平均检索长度$ASL_{SS}=(s+n/s)/2+1$。

所以当$S=\sqrt{n}$时，ASL_{SS}取得最小值$\sqrt{n}+1$。

答案：在分块检索中，对256个元素的线性表分为 16 块最好，每块的最佳长度是 17 ；若每块的长度为8，其平均检索长度为 21 。

【自测题9-13】

采用中序遍历即可。

```
TYPE PNODE=^NODE;
     NODE=RECORD
     data:       INTEGER;
     lchild，rchild:PNODE;
END;
```

```
PROCEDURE display(T:PNODE;x:integer);
BEGIN
    IF T<>NIL THEN BEGIN
        display(T^.lchild，x);
        IF T^.data>=x THEN WRITE(T^.data);
        display(T^.rchild，x);
    END;
END;
```

【自测题9-14】

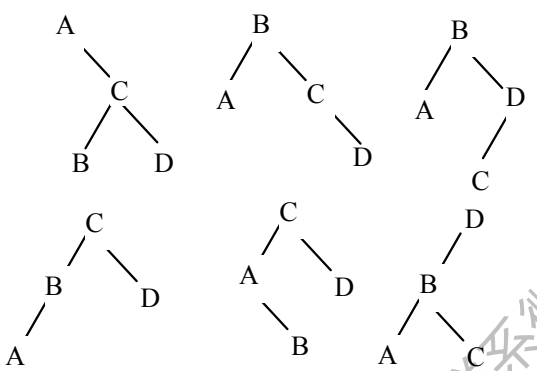
① 错误。最佳二叉树是静态树表，AVL是动态树表，二者范围不同。

② 正确。可以用反证法求证。若该中序后继结点*p*有两个子女，则左子女是*p*的中序前继结点，则*p*不是中序后继，矛盾。

③ 错误。α越小，只能说发生冲突的可能性越小，但依然有可以发生冲突。

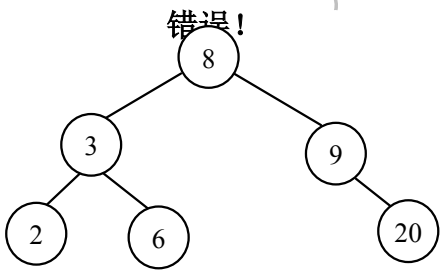
【自测题9-15】

不同的二叉排序树如下图所示。

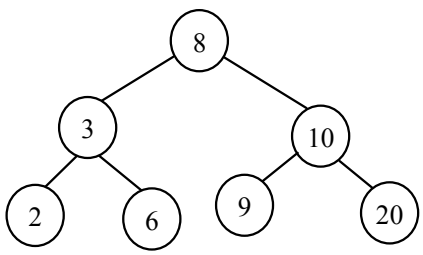


【自测题9-16】

插入关键码为6的结点后，AVL树如下图所示。



再插入关键码为10的结点后，AVL树如下图所示。

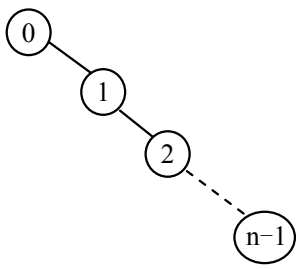


【自测题9-17】

有序表顺序查找的算法：

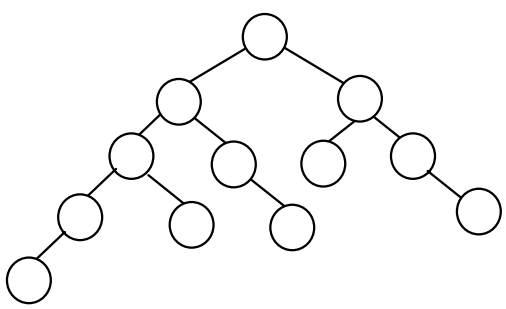
```
int Search(STable st，KeyType key){
    //在有序顺序表中顺序查找关键字为key的元素，
    //若找到，返回该元素在表中的位置，否则返回-1
    for (i=0;i<st.length;i++) {
        if (st.elem[i].key==key) return I;
        else if (st.elem[i].key>key) return-1;
    }
    return-1;
}
```

顺序查找的判定树如下图所示。



【自测题9-18】

最大深度是5，如下图所示。



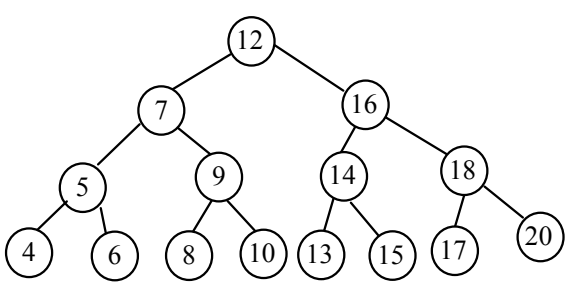
【自测题9-19】

平均查找长度=$\frac{1}{n}\sum_{i=1}^{n}h_i$，$h_i$为结点$i$的深度。因此要使平均查找长度最小，就要使结点深度和最小，所以应该为完全二叉树。有15个关键字，所以构造一个高度为4的满二叉树刚好符合条件。

将关键字从小到大排序，根结点的值为排序后的中间元素，即第8个元素。

依次类推即可。

①最佳二叉树如下图所示。



② 优化算法如下：

```
PROCEDURE bfind(p:bptr;K:keytype);
{在指针p所指的二叉排序树上查找其关键字等于给定值K的记录，当查找成功时返回所找结点的
指针}
BEGIN
      IF p^.key=k
          THEN return(p);
      ELSE IF K<>p^.key
          THEN IF b^.Lc<>NIL THEN bfind(p^.Lc，K);
      ELSE return(NIL)
      ELSE IF P^.Rc<>NIL THEN bfind(p^.Rc，K);
      ELSE return(NIL);
END
```

【自测题9-20】

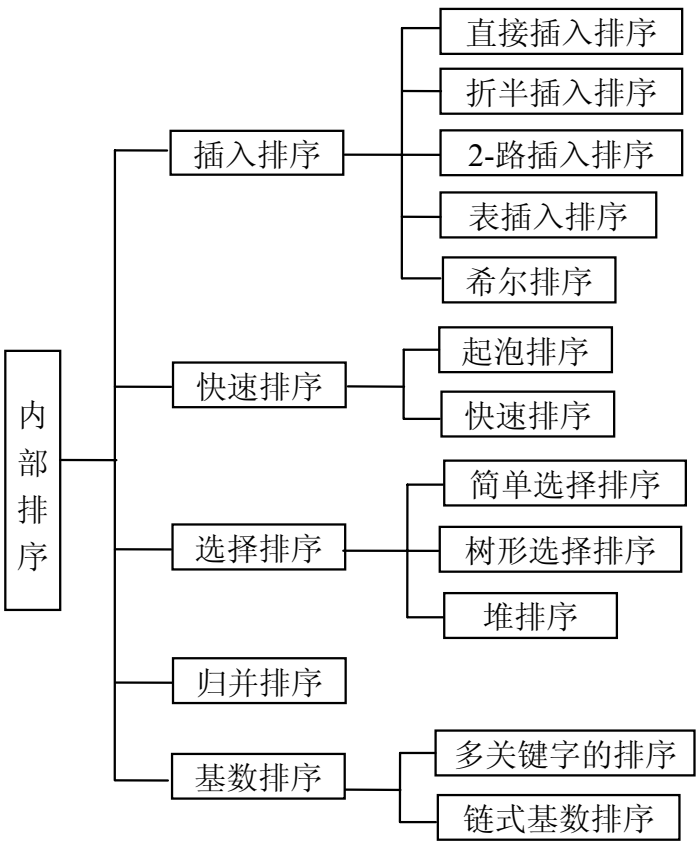
① 字符树中从根到一个树叶的路径上的所有结点连接起来构成（字符串）。

② 在分块检索中，对256个元素的线性表分为（16）块最好，每块的最佳长度为（16）；若每块的长度为8，其平均检索长度为（21）。

③ 静态索引结构是指在系统运行时（索引表）不会发生改变。

第 10 章　内 部 排 序

10.1　基本知识结构图



10.2　知 识 点

1. 排序

排序是将一个数据元素（或记录）的任意序列，重新排列成一个按关键字有序的序列。排序的方法有很多种，按照排序的稳定性分类，可分为稳定的排序方法和不稳定的排序方法；按照排序过程中涉及的存储器不同，可将排序方法分为内部排序和外部排序；按照排序过程中依据的不同原则对内部排序方法进行分类，可分为插入排序、交换排序、选择排序、归并排序和计数排序；按照内部排序过程中所需的工作量来区分，可分为简单的排序

方法、先进的排序方法和基数排序。

2. 插入排序

插入排序是将一个记录插入到一排好序的有序表中，从而得到一个新的、记录数增1的有序表。主要有：直接插入排序、折半插入排序、2-路插入排序、表插入排序和希尔排序。

由于插入排序的基本操作是在一个有序表中进行查找和插入，其中“查找”操作可利用“折半查找”来实现，由此进行的插入排序称之为折半插入排序。

2-路插入排序是在折半插入排序的基础上再改进之，其目的是减少排序过程中移动记录的次数，但为此需要n个记录的辅助空间。

表插入排序：首先将静态链表中数组下标为“1”的分量（结点）和表头结点构成一个循环链表，然后依次将下标为“2”至“n”的分量（结点）按记录关键字非递减有序插入到循环链表中。

希尔排序又称“缩小增量排序”，它的基本思想是：先将整个待排记录序列分割成为若干子序列分别进行直接插入排序，待整个序列中的记录“基本有序”时，再对全体记录进行一次直接插入排序。

3．快速排序

快速排序是一种借助“交换”进行排序的方法。最简单的一种借助交换进行排序的方法是起泡排序。快速排序是对起泡排序的一种改进，其平均时间复杂度为$O(n\log_n)$。就平均时间而言，快速排序是目前被认为最好的一种内部排序方法。

4. 选择排序

选择排序的基本思想是：每一趟在$n-i+1$(i=1，2，…，n-1）个记录选取关键字最小的记录作为有序序列中第i个记录。有简单选择排序、树形选择排序和堆排序三种。

5. 归并排序

归并排序的含义是将两个或两个以上的有序表组合成一个新的有序表。与快速排序和堆排序相比，归并排序的最大特点是，它是一种稳定的排序方法。

6. 基数排序

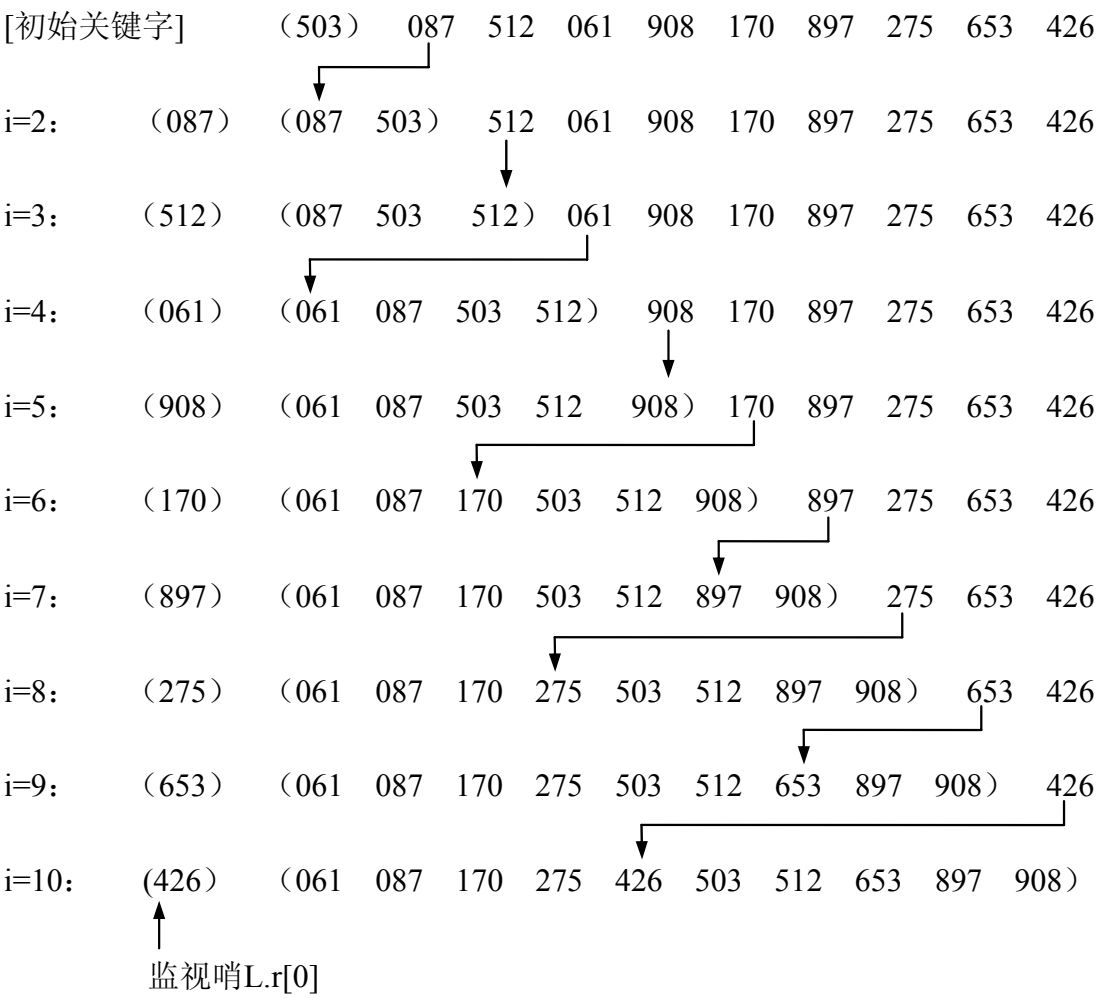
基数排序是一种借助多关键字排序的思想对单逻辑关键字进行排序的方法，它适合于n值很大而关键字较小的序列。

10.3　习题及参考答案

1. 以关键码序列（503，087，512，061，908，170，897，275，653，426）为例，手工执行以下排序算法，写出每一趟排序结束时的关键码状态：

（1）直接插入排序；　　（2）希尔排序（增量d[1]=5）；
（3）快速排序；　　（4）堆排序；
（5）归并排序；　　（6）基数排序。

【解答】（1）直接插入排序：



（2）希尔排序（增量d[1]=5）：

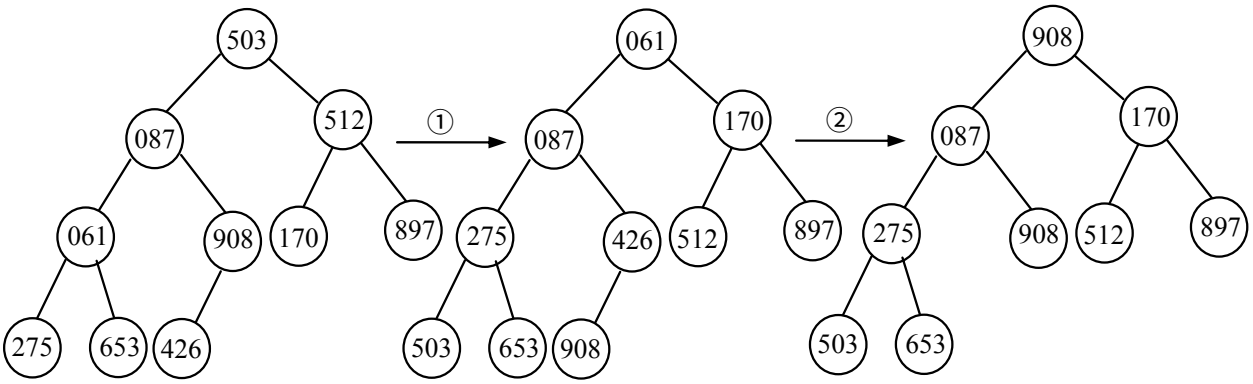
[初始关键字]	503	087	512	061	908	170	897	275	653	426
一趟排序结果：	170	087	275	061	426	503	897	512	653	908
二趟排序结果：	061	087	275	170	426	503	897	512	653	908
三趟排序结果：	061	087	170	275	426	503	512	653	897	908

（3）快速排序：

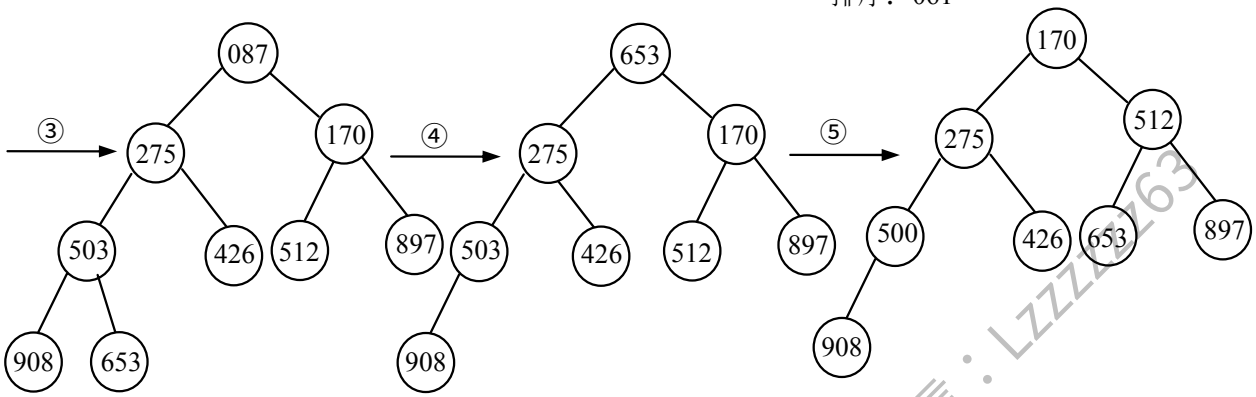
初始关键字：	503	087	512	061	908	170	897	275	653	426
第一趟排序后：	087	503	061	512	170	897	275	653	426	908
第二趟排序后：	087	061	503	170	512	275	653	426	897	
第三趟排序后：	061	087	170	503	275	512	426	653		
第四趟排序后：	061	087	170	275	503	426	512			
第五趟排序后：	061	087	170	275	426	503				

第六趟排序后：　061　087　170　275　426

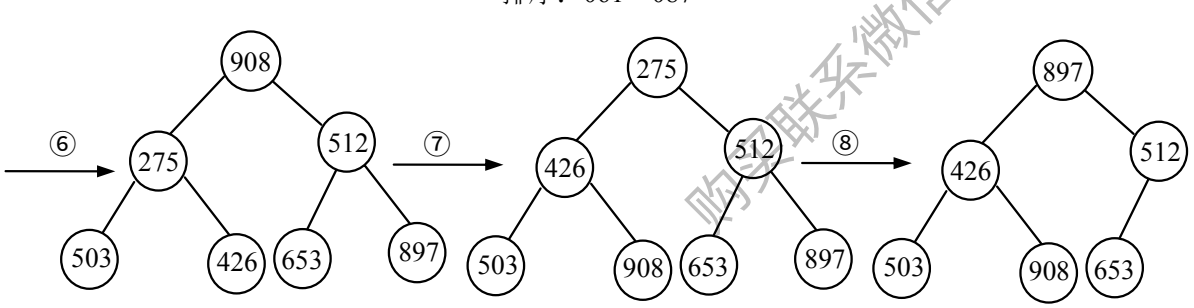
（4）堆排序：



排序：061

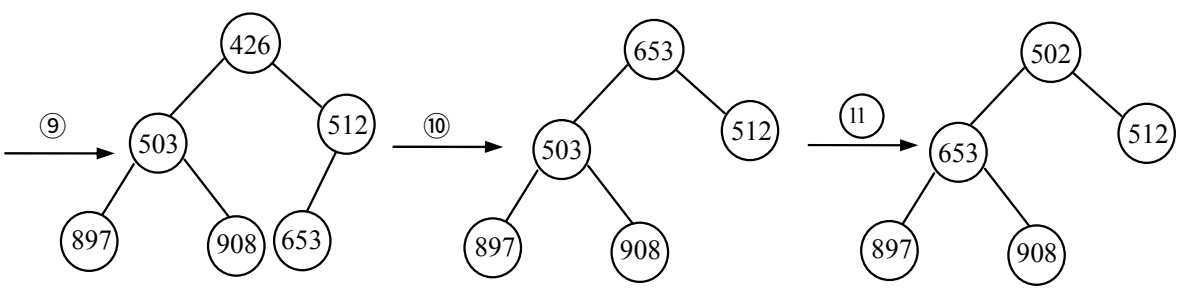


排序：061　087

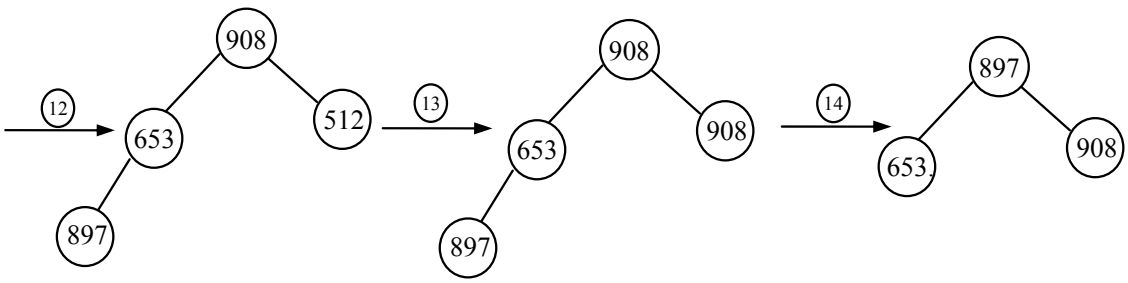


排序：061　087　170

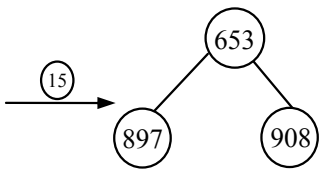
排序：061　087　170　275



排序：061　087　170　275　426



排序：061 087 170 275 426 502　　排序：061 087 170 275 426 502 512

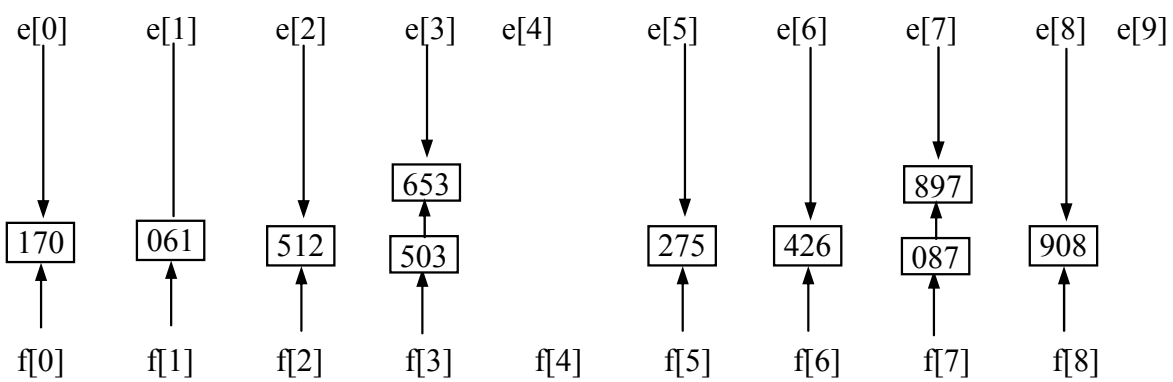


排序：061 087 275 426 502 512 653 897 908

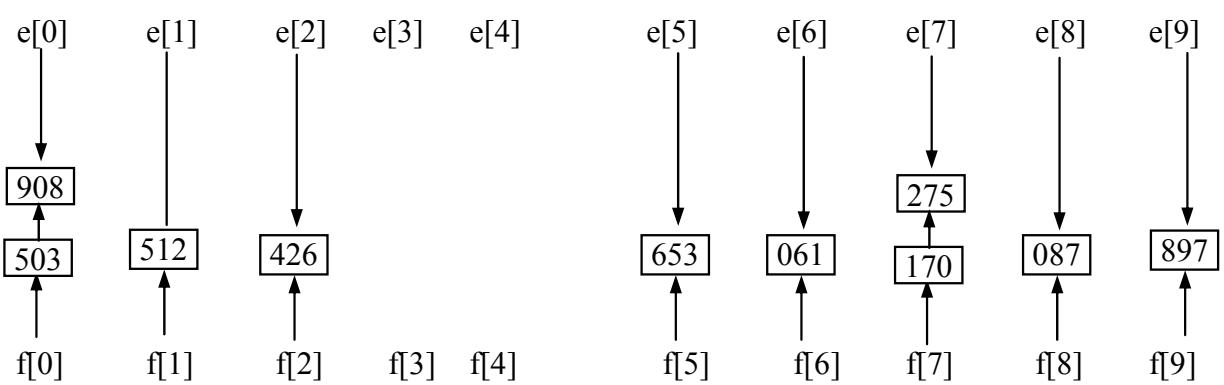
（5）归并排序：

初始关键字：	503	087	512	061	908	170	897	275	653	426
一趟归并之后：	087	503	061	512	170	908	275	897	426	653
二趟归并之后：	061	087	503	512	170	275	897	908	426	653
三趟归并之后：	061	087	170	275	503	512	897	908	426	653
四趟归并之后：	061	087	170	275	426	503	512	653	897	908

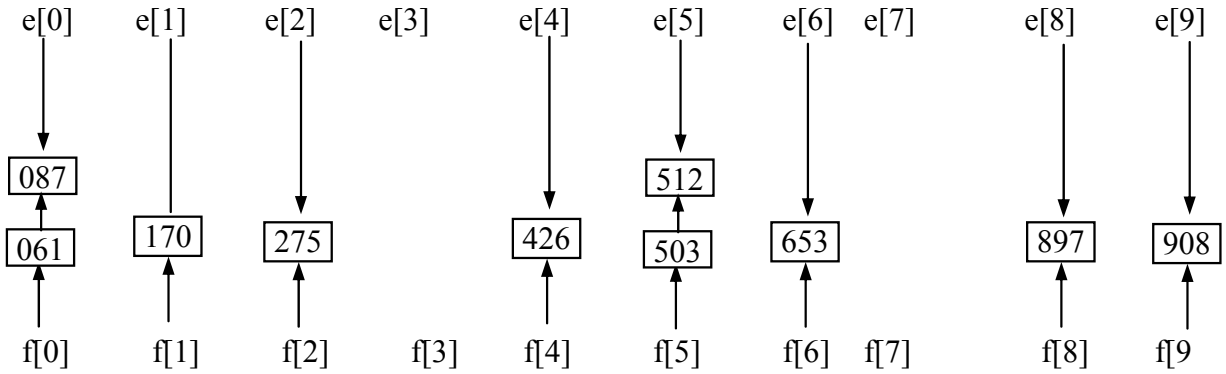
（6）基数排序：



第一趟收集之后：170→061→512→503→653→275→426→087→897→908



第二趟收集之后：503→908→512→426→653→061→170→275→087→897



第三趟收集之后：061→087→170→275→426→503→512→653→897→908

2．若对下列关键字序列进行快速排序和归并排序，分别写出三次调用过程Partition和过程Merge后的结果。

（Tim， Kay， Eva， Roy， Dot， Jon， Kim， Ann， Tom， Jim， Guy， Amy）

【解答】三次调用过程Partition的结果分别为

（Amy， Kay， Eva， Roy， Dot， Jon， Kim， Ann， Guy， Jim）Tim（Tom）

Amy（Kay， Eva， Roy， Dot， Jon， Kim， Ann， Guy， Jim）Tim（Tom）

Amy（Jim， Eva， Guy， Dot， Jon， Ann）Kay（Kim， Roy）Tim（Tom）

三次调用过程Merge的结果分别为

（Kay， Tim， Eva， Roy， Dot， Jon， Kim， Ann， Tom， Jim， Guy， Amy）

（Eva， Kay， Tim， Roy， Dot， Jon， Kim， Ann， Tom， Jim， Guy， Amy）

（Eva， Kay， Tim， Dot， Roy， Jon， Kim， Ann， Tom， Jim， Guy， Amy）

3．试问在直接插入排序、快速排序、归并排序、希尔排序、堆排序和基数排序六种排序方法中，哪些是稳定的？哪些是不稳定的？并为每一种不稳定的排序方法举出一个不稳定的实例。

【解答】希尔排序、快速排序和堆排序是不稳定的排序方法。

4．试问：对初始状态如下（长度为n）的各序列进行直接插入排序时，至多需进行多少次关键字间的比较（要求排序后的序列按关键字自小至大顺序有序）？

（1）关键字（自小至大）顺序有序；（$key_1<key_2<\cdots<key_n$）。

（2）关键字（自大至小）逆序有序；（$key_1>key_2>\cdots>key_n$）。

（3）序号为奇数的关键字顺序有序，序号为偶数的关键字顺序有序；

（$key_1<key_2<\cdots$，　$key_2<key_4\cdots$）

（4）前半个序列中的关键字顺序有序，后半个序列中的关键字逆序有序：

（$key_1<key_2<\cdots<key_{|n/2|}$，　$key_{|n/2|+1}>\cdots key_n$）

【解答】（1）0

（2）$\dfrac{n(n-1)}{2}$

（3）$n-1+\sum_{i=1}^{\left\lceil \frac{n}{2} \right\rceil} i$

（4）$\left\lfloor \frac{n}{2} \right\rfloor -1+\sum_{i=\left\lceil \frac{n}{2} \right\rceil +1}^{n} i$

5．假设我们把n个元素的序列$\{a_1, a_2, \cdots, a_n\}$中满足条件$a_k < \max_{1\le t<k}\{a_t\}$的元素$a_k$称为“逆序元素”。若在一个无序序列中有一对元素$a_i>a_j(i<j)$，试问当将$a_i$和$a_j$相互交换之后（即序列由$\{\cdots a_i \cdots a_j \cdots\}$变为$\{\cdots a_j \cdots a_i \cdots\}$），该序列中逆元素的个数会有什么变化？为什么？

【解答】只要考虑序列中位于a_i和a_j之间的元素$a_k(i<k<j)$的几种情况即可，当$a_k>a_i>a_j$时，有可能使a_i由非逆序元素变为逆序元素，但整个序列的“逆序对”不变。

6．奇偶交换排序如下所述：第一趟对所有奇数i，将$a[i]$和$a[i+1]$进行比较；第二趟对所有的偶数i，将$a[i]$和$a[i+1]$进行比较，若$a[i]>a[i+1]$，则将两者交换；第三趟对奇数i；第四趟对偶数i，…，依次类推直至整个序列有序为止。

（1）试问这种排序方法的结束条件是什么？

（2）分析当初始序列为正序或逆序两种情况下，奇偶交换排序过程中所需进行的关键字比较的次数。

【解答】（1）连续两趟对所有奇数进行一次比较和对所有偶数进行一次比较，都不需交换，则排序结束。

（2）正序情况下，需比较$2\left\lfloor \frac{n}{2} \right\rfloor$次，逆序情况下需比较$(n+1)\left\lfloor \frac{n}{2} \right\rfloor$。

7．不难看出，对长度为n的记录序列进行快速排序时，所需进行的比较次数依赖于这n个元素的初始排列。

（1）n=7时在最好情况下需进行多少次比较？请说明理由。

（2）对n=7给出一个最好情况的初始排列实例。

【解答】（1）快速排序的最好情况是指，排序所需的“关键字间的比较次数”和“记录的移动次数”最少的情况，在n=7时，至少需进行2趟排序。

（2）初始排列为4 1 3 2 6 5 7时，只需进行10次比较，2趟排序即可完成，即为最好情况。

8．一个长度为n的序列，若去掉其中少数k（$k<<n$）个记录后，序列是按关键字有序的，则称为近似有序序列。试对这种序列讨论各种简单排序方法的时间复杂度。

【解答】对近似有序的序列可用直接插入排序法，其时间复杂度为$O(k \cdot n)$，当$k<<n$且n很大时，将比快速排序更有效。而起泡排序和简单选择排序的时间复杂度均为$O(n^2)$，和k无关。本题旨在提醒读者注意算法效率讨论的前提条件。

9．试以L.r[k+1]作为监视哨改写直接插入排序算法。其中，L.r[1..k]为待排序记录且k<MAXSIZE。

【解答】算法如下：

```
void Insert_Sort1(SqList &L)//监视哨设在高下标端的插入排序算法
{
   k=L.length;
   for(i=k-1;i;--i) //从后向前逐个插入排序
      if(L.r[i].key>L.r[i+1].key)
      {
         L.r[k+1].key=L.r[i].key; //监视哨
         for(j=i+1;L.r[j].key>L.r[i].key;++j)
            L.r[j-1].key=L.r[j].key; //前移
         L.r[j-1].key=L.r[k+1].key; //插入
      }
}//Insert_Sort1
```

10. 试编写2-路插入排序的算法。

【解答】算法的核心语句为

```
if (r[i].key<d[1].key) {
   if (first ==1) {d[n]=r[i]; first=n; }
   else {
      binpass (d，  first，  n，  r[i].key，  k);          {查找插入位置}
      d[first-1..k-1]=d[first..k];
      d[k]=r[i]; first - -;
      } // else
   } // if
else …

void binpass (RcdType d[ ]，  int l，  int h，  keytype x，  int &m)
{
      // 在d[1..h]中折半查找x的插入位置m，
      // 要求d[m].key<=x<d[m+1].key。
      while (<=h) {
            m= (l+h) /2;
            if (x<d[m].key) h=m-1;
            else l= m+1;
        }
        m=h;
   } // binpass
```

11．试编写链表插入排序的算法。

【解答】算法如下：

```
void SLInsert_Sort(SLList &L)//静态链表的插入排序算法
{
      L.r[0].key=0;L.r[0].next=1;
      L.r[1].next=0; //建初始循环链表
      for(i=2;i<=L.length;i++) //逐个插入
      {
            p=0;x=L.r[i].key;
            while(L.r[L.r[p].next].key<x&&L.r[p].next)
```

```
            p=L.r[p].next;
        q=L.r[p].next;
        L.r[p].next=i;
        L.r[i].next=q;
    }//for
    p=L.r[0].next;
    for(i=1;i<L.length;i++) //重排记录的位置
    {
        while(p<i) p=L.r[p].next;
        q=L.r[p].next;
        if(p!=i)
        {
            L.r[p]<->L.r[i];
            L.r[i].next=p;
        }
        p=q;
    }//for
}//SLInsert_Sort
```

12．编写一个双向起泡的排序算法，即相邻两遍向相反方向起泡。

【解答】算法如下：

```
void Bubble_Sort2(int a[ ]，int n)//相邻两遍是反方向起泡的冒泡排序算法
{
  low=0;high=n-1; //冒泡的上下界
  change=1;
  while(low<high&&change)
  {
    change=0;
    for(i=low;i<high;i++) //从上向下起泡
      if(a[i]>a[i+1])
      {
        a[i]<->a[i+1];
        change=1;
      }
    high--; //修改上界
    for(i=high;i>low;i--) //从下向上起泡
      if(a[i]<a[i-1])
      {
        a[i]<->a[i-1];
        change=1;
      }
    low++; //修改下界
  }//while
}//Bubble_Sort2
```

13．修改12题中要求的算法，请考虑如何避免将算法写成两个并在一起的相似的单向起泡的算法段。

【分析】这是一个技巧性很强的程序设计练习，旨在培养综合能力。但这样的代码可

读性较差，故在实际应用中并不鼓励这样做。

【解答】算法如下：

```
void BiBubbleSort (SqList)
{
    // L存待排序文件，返回时它有序。
    d=1;    // d为增量，正向和反向起泡时值分别为1和−1，初值为1（正向）
    pos[0]=1;   pos[2]=n;          // pos[d+1]为本趟扫描的终止下标
    i=1;   exchanged=TRUE;    // 记录一趟扫描中有否交换
    while (exchanged)
    {          // 扫描一趟
        exchanged=FALSE;          // 本趟初始无交换
        while (i!=pos[d+1])
        {
            if ((L.r[i].key−L.r[I+d].key) *d>0)
            {
                L.r[i] <−>L.r[i+d];   exchanged=TRUE;
            }
                i+=d;
        }
        pos[d+1]−= d;   i=pos[d+1];   d=−d;   // 转向
       }
} // BiBubbleSort
```

14．编写奇偶交换排序的算法。

【解答】算法如下：

```
void OE_Sort(int a[ ]，int n)//奇偶交换排序的算法
{
  change=1;
  while(change)
  {
    change=0;
    for(i=1;i<n−1;i+=2) //对所有奇数进行一趟比较
      if(a[i]>a[i+1])
      {
        a[i]<−>a[i+1];
        change=1;
      }
    for(i=0;i<n−1;i+=2) //对所有偶数进行一趟比较
      if(a[i]>a[i+1])
      {
        a[i]<−>a[i+1];
        change=1;
      }
  }//while
}//OE_Sort
```

15．按下述原则编写快排的非递归算法；

（1）一趟排序之后，先对长度较短的子序列进行排序，且将另一子序列的上、下界入栈保存；

（2）若待排记录数小于等于3，则不再进行分割，而是直接进行比较排序。

【解答】算法如下：

```
typedef struct {
                    int low;
                    int high;
                } boundary; //子序列的上下界类型
void QSort_NotRecurve(int SQList &L)//快速排序的非递归算法
{
  low=1;high=L.length;
  InitStack(S); //S的元素为boundary类型
  while(low<high&&!StackEmpty(S)) //注意排序结束的条件
  {
    if(high-low>2) //如果当前子序列长度大于3且尚未排好序
    {
      pivot=Partition(L，low，high); //进行一趟划分
      if(high-pivot>pivot-low)
      {
        Push(S，{pivot+1，high}); //把长的子序列边界入栈
        high=pivot-1; //短的子序列留待下次排序
      }
      else
      {
        Push(S，{low，pivot-1});
        low=pivot+1;
      }
    }//if
    else if(low<high&&high-low<3)//如果当前子序列长度小于3且尚未排好序
        {
            Easy_Sort(L，low，high); //直接进行比较排序
            low=high; //当前子序列标志为已排好序
        }
        else //如果当前子序列已排好序但栈中还有未排序的子序列
        {
            Pop(S，a); //从栈中取出一个子序列
            low=a.low;
            high=a.high;
        }
  }//while
}//QSort_NotRecurve
int Partition(SQList &L，int low，int high)//一趟划分的算法
{
  L.r[0]=L.r[low];
  pivotkey=L.r[low].key;
  while(low<high)
  {
    while(low<high&&L.r[high].key>=pivotkey)
      high--;
```

```
    L.r[low]=L.r[high];
    while(low<high&&L.r[low].key<=pivotkey)
      low++;
    L.r[high]=L.r[low];
  }//while
  L.r[low]=L.r[0];
  return low;
}//Partition
void Easy_Sort(SQList &L，int low，int high)//对长度小于3的子序列进行比较排序
{
  if(high−low==1) //子序列只含两个元素
    if(L.r[low].key>L.r[high].key) L.r[low]<−>L.r[high];
  else //子序列含有三个元素
  {
    if(L.r[low].key>L.r[low+1].key) L.r[low]<−>L.r[low+1];
    if(L.r[low+1].key>L.r[high].key) L.r[low+1]<−>L.r[high];
    if(L.r[low].key>L.r[low+1].key) L.r[low]<−>L.r[low+1];
  }
}//Easy_Sort
```

16．编写算法，对n个关键字取整数值的记录序列进行整理，以使所有关键字为负值的记录排在关键字为非负值的记录之前，要求：

（1）采用顺序存储结构，至多使用一个记录的辅助存储空间；

（2）算法的时间复杂度为$O(n)$；

（3）讨论算法中记录的最大移动次数。

【解答】算法如下：

```
void Divide(int a[ ]，int n)//把数组a中所有值为负的记录调到非负的记录之前
{
  low=0;high=n−1;
  while(low<high)
  {
    while(low<high&&a[high]>=0) high−−; //以0作为虚拟的枢轴记录
    a[low]<−>a[high];
    while(low<high&&a[low]<0) low++;
    a[low]<−>a[high];
  }
}//Divide
```

算法中记录的最大移动次数为$n/2$。

17．荷兰国旗问题：设有一个仅由红、白、蓝三种颜色的条块组成的条块序列。请编写一个时间复杂度为$O(n)$的算法，使得这些条块按红、白、蓝的顺序排好，即排成荷兰国旗图案。

【分析】在算法中设立三个指针，其中，j表示当前元素，i以前的元素全部为红色，k以后的元素全部为蓝色，这样，就可以根据j的颜色，把其交换到序列的前部或者后部。

【解答】算法如下：

```
typedef enum {RED，WHITE，BLUE} color; //三种颜色
void Flag_Arrange(color a[ ]，int n)//把由三种颜色组成的序列重排为按照红，白，蓝的顺序排列
{
  i=0;j=0;k=n-1;
  while(j<=k)
    switch(a[j])
    {
      case RED:
        a[i]<->a[j];
        i++;
        j++;
        break;
      case WHITE:
        j++;
        break;
      case BLUE:
        a[j]<->a[k];
        k--; //这里没有j++;语句是为了防止交换后a[j]仍为蓝色的情况
    }
}//Flag_Arrange
```

18．试以单链表为存储结构实现简单选择排序的算法。

【解答】算法如下：

```
void LinkedList_Select_Sort(LinkedList &L)//单链表上的简单选择排序算法
{
  for(p=L;p->next->next;p=p->next)
  {
    q=p->next;x=q->data;
    for(r=q，s=q;r->next;r=r->next) //在q后面寻找元素值最小的结点
      if(r->next->data<x)
      {
        x=r->next->data;
        s=r;
      }
    if(s!=q) //找到了值比q->data更小的最小结点s->next
    {
      p->next=s->next;s->next=q;
      t=q->next;q->next=p->next->next;
      p->next->next=t;
    } //交换q和s->next两个结点
  }//for
}//LinkedList_Select_Sort
```

19．已知（k_1，k_2，…，k_p）为堆，则可以写一个时间复杂度$O(\log n)$的算法将（k_1，k_2，…，k_p，k_{p+1}）调整为堆。试编写“从p=1起，逐个插入建堆”的算法。

【解答】算法如下：

```
void Build_Heap(Heap &H，int n)//从低下标到高下标逐个插入建堆的算法
{
```

```
  for(i=2;i<n;i++)
  { //此时从H.r[1]到H.r[i-1]已经是大顶堆
    j=i;
    while(j!=1) //把H.r[i]插入
    {
      k=j/2;
      if(H.r[j].key>H.r[k].key)
        H.r[j]<->H.r[k];
      j=k;
    }
  }//for
}//Build_Heap
```

20．假设定义堆为满足如下性质的完全三叉树：（1）空树为堆；（2）根结点的值不小于所有子树根的值，且所有子树均为堆。编写利用上述定义的堆进行排序的算法。

【分析】在此堆排序算法中：① 建初始堆时，*i*的上限从H.length/3开始；②调整堆的时候，要从结点的三个孩子结点中选择最大的那一个，最左边的孩子的序号的计算公式为j=3*(s-1)。

【解答】算法如下：

```
void TriHeap_Sort(Heap &H)//利用三叉树形式的堆进行排序的算法
{
  for(i=H.length/3;i>0;i--)
    Heap_Adjust(H，i，H.length);
  for(i=H.length;i>1;i--)
  {
    H.r[1]<->H.r[i];
    Heap_Adjust(H，1，i-1);
  }
}//TriHeap_Sort
void Heap_Adjust(Heap &H，int s，int m)//顺序表H中，H.r[s+1]到H.r[m]已经是堆，把H.r[s]插入并调
                                       //整成堆
{
  rc=H.r[s];
  for(j=3*s-1;j<=m;j=3*j-1)
  {
    if(j<m&&H.r[j].key<H.r[j+1].key) j++;
    H.r[s]=H.r[j];
    s=j;
  }
  H.r[s]=rc;
}//Heap_Adjust
```

21．可按如下所述实现归并推序：假设序列中有*k*个长度为≤*l*的有序子序列。利用过程merge对它们进行两两归并，得到[*k*/2]个长度≤2*l*的有序子序列，称为一趟归并排序。反复调用一趟归并排序过程，使有序子序列的长度自*l*=1开始成倍地增加，直至使整个序列成为一个有序序列。试对序列实现上述归并排序的递推算法。

【解答】算法如下：

```
void Merge_Sort(int a[ ]，int n)//归并排序的非递归算法
{
  for(l=1;l<n;l*=2) //l为一趟归并段的段长
    for(i=0;(2*i-1)*l<n;i++) //i为本趟的归并段序号
    {
      start1=2*l*i; //求出待归并的两段的上下界
      end1=start1+l-1;
      start2=end1+1;
      end2=(start2+l-1)>(n-1)?(n-1):(start2+l-1);//注意end2可能超出边界
      Merge(a，start1，end1，start2，end2); //归并
    }
}//Merge_Sort
void Merge(int a[ ]，int s1，int e1，int s2，int e2)//将有序子序列a[s1]到a[e1]和a[s2]到a[e2]归并为有序
                                                    //序列a[s1]到a[e2]
{
  int b[MAXSIZE]; //设立辅助存储数组b
  for(i=s1，j=s2，k=s1;i<=e1&&j<=e2;k++)
  {
    if(a[i]<a[j]) b[k]=a[i++];
    else b[k]=a[j++];
  }
  while(i<=e1) b[k++]=a[i++];
  while(j<=e2) b[k++]=a[j++]; //归并到b中
  for(i=s1;i<=e2;i++) //复制回去
    a[i]=b[i];
}//Merge
```

22．采用链表存储结构，重做21题。

【解答】算法如下：

```
void LinkedList_Merge_Sort1(LinkedList &L)//链表结构上的归并排序非递归算法
{
  for(l=1;l<L.length;l*=2) //l为一趟归并段的段长
    for(p=L->next，e2=p;p->next;p=e2)
    {
      for(i=1，q=p;i<=l&&q->next;i++，q=q->next);
      e1=q;
      for(i=1;i<=l&&q->next;i++，q=q->next);
      e2=q; //求出两个待归并子序列的尾指针
      if(e1!=e2) LinkedList_Merge(L，p，e1，e2); //归并
    }
}//LinkedList_Merge_Sort1
void LinkedList_Merge(LinkedList &L，LNode *p，LNode *e1，LNode *e2)
     //对链表上的子序列进行归并，第一个子序列是从p->next到e1，第二个是从e1->next到e2
{
  q=p->next;r=e1->next; //q和r为两个子序列的起始位置
  while(q!=e1->next&&r!=e2->next)
  {
    if(q->data<r->data) //选择关键字较小的那个结点接在p的后面
```

```
    {
      p->next=q;p=q;
      q=q->next;
    }
    else
    {
      p->next=r;p=r;
      r=r->next;
    }
  }//while
  while(q!=e1->next) //接上剩余部分
  {
    p->next=q;p=q;
    q=q->next;
  }
  while(r!=e2->next)
  {
    p->next=r;p=r;
    r=r->next;
  }
}//LinkedList_Merge
```

23．2-路归并排序的另一策略是，先对待排序序列扫描一遍，找出并划分为若干个最大有序子列，将这些子列作为初始归并段。试写一个算法在链表结构上实现这一策略。

【分析】可附设循环队列，将一遍所得各有序子列的头指针存入队列，然后令队头的两个指针出列，且将这两个链序列归并后的新链序列的头指针入列，如此重复直至队列中只有一个链序列的头指针为止。

【解答】算法如下：

```
void LinkedList_Merge_Sort2(LinkedList &L)//初始归并段为最大有序子序列的归并排序，采用链表存
                                          //储结构
{
  LNode *end[MAXSIZE]; //设立一个数组来存储各有序子序列的尾指针
  for(p=L->next->next，i=0;p;p=p->next) //求各有序子序列的尾指针
    if(!p->next| |p->data>p->next->data) end[i++]=p;
  while(end[0]->next) //当不止一个子序列时进行两两归并
  {
    j=0;k=0; //j:当前子序列尾指针存储位置;k:归并后的子序列尾指针存储位置
    for(p=L->next，e2=p;p->next;p=e2) //两两归并所有子序列
    {
      e1=end[j];e2=end[j+1]; //确定两个子序列
      if(e1->next) LinkedList_Merge(L，p，e1，e2); //归并
      end[k++]=e2; //用新序列的尾指针取代原来的尾指针
      j+=2; //转到后面两个子序列
    }
  }//while
}//LinkedList_Merge_Sort2
void LinkedList_Merge(LinkedList &L，LNode *p，LNode *e1，LNode *e2)
     //对链表上的子序列进行归并，第一个子序列是从p->next到e1，第二个是从e1->next到e2
```

```
{
  q=p->next;r=e1->next;
  while(q!=e1->next&&r!=e2->next)
  {
    if(q->data<r->data)
    {
      p->next=q;p=q;
      q=q->next;
    }
    else
    {
      p->next=r;p=r;
      r=r->next;
    }
  }//while
  while(q!=e1->next)
  {
    p->next=q;p=q;
    q=q->next;
  }
  while(r!=e2->next)
  {
    p->next=r;p=r;
    r=r->next;
  }
}//LinkedList_Merge
```

24．已知两个有序序列（a_1，a_2，…，a_m）和（a_{m+1}，a_{m+2}，…，a_n），并且其中一个序列的记录个数少于s，且s=$\lfloor\sqrt{n}\rfloor$。试写一个算法，用$O(n)$时间和$O(1)$附加空间完成这两个有序序列的归并。

【分析】如果第一个序列的记录个数少于s，则确定其第一个记录在归并后的序列中的位置k，并从该位置开始进行$k-1$个位置的循环移位。这次循环移位仅涉及第一个序列中的记录和第二个序列中的前$k-1$个记录，使得前k个记录就位于其归并后的最终位置。如此继续下去，依次对原来第一个序列的第二个及随后的每个记录重复上述过程。

【解答】算法如下：

```
void SL_Merge(int a[ ]，int l1，int l2)//把长度分别为l1，l2且l1^2<(l1+l2)的两个有序子序列归并为有
                                       //序序列
{
  start1=0;start2=l1; //分别表示序列1和序列2的剩余未归并部分的起始位置
  for(i=0;i<l1;i++) //插入第i个元素
  {
    for(j=start2;j<l1+l2&&a[j]<a[start1+i];j++); //寻找插入位置
    k=j-start2; //k为要向右循环移动的位数
    RSh(a，start1，j-1，k);//将a[start1]到a[j-1]之间的子序列循环右移k位
    start1+=k+1;
    start2=j; //修改两序列尚未归并部分的起始位置
```

```
    }
}//SL_Merge
void RSh(int a[ ]，int start，int end，int k)//将a[start]到a[end]之间的子序列循环右移k位
{
    len=end-start+1;
    for(i=1;i<=k;i++)
        if(len%i==0&&k%i==0) p=i; //求len和k的最大公约数p
    for(i=0;i<p;i++) //对p个循环链分别进行右移
    {
        j=start+i;l=start+(i+k)%len;temp=a[j];
        while(l!=start+i)
        {
            a[j]=temp;
            temp=a[l];
            a[l]=a[j];
            j=l;l=start+(j-start+k)%len; //依次向右移
        }
        a[start+i]=temp;
    }//for
}//RSh
```

25．假设有1000个关键字为小于10000的整数的记录序列，请设计一种排序方法，要求以尽可能少的比较次数和移动次数实现排序，并按你的设计编出算法。

【解答】利用哈希表进行排序。

```
void Hash_Sort(int a[ ])//对1000个关键字为四位整数的记录进行排序
{
    int b[10000];
    for(i=0;i<1000;i++) //直接按关键字散列
    {
        for(j=a[i];b[j];j=(j+1)%10000);
        b[j]=a[i];
    }
    for(i=0，j=0;i<1000;j++) //将散列收回a中
        if(b[j])
        {
            for(x=b[j]，k=j;b[k];k=(k+1)%10000)
                if(b[k]==x)
                {
                    a[i++]=x;
                    b[k]=0;
                }
        }//if
}//Hash_Sort
```

26．序列的“中值记录”指的是：如果将此序列排序后，它是第$\left\lceil \frac{n}{2} \right\rceil$个记录。试写一个求中值记录的算法。

【解答】算法如下：

```
typedef struct {
                    int gt; //大于该记录的个数
                    int lt; //小于该记录的个数
                } place; //整个序列中比某个关键字大或小的记录个数
int Get_Mid(int a[ ]，int n)//求一个序列的中值记录的位置
{
  place b[MAXSIZE];
  for(i=0;i<n;i++) //对每一个元素统计比它大和比它小的元素个数gt和lt
    for(j=0;j<n;j++)
    {
      if(a[j]>a[i]) b[i].gt++;
      else if(a[j]<a[i]) b[i].lt++;
    }
  mid=0;
  min_dif=abs(b[0].gt-b[0].lt);
  for(i=0;i<n;i++) //找出gt值与lt值最接近的元素，即为中值记录
    if(abs(b[i].gt-b[i].lt)<min_dif) mid=i;
  return mid;
}//Get_Mid
```

27．已知记录序列a[1..n]中的关键字各不相同，可按如下所述实现计数排序：另设数组c[1..n]，对每个记录a[i]，统计序列中关键字比它小的记录个数存于c[i]，则c[i]=0的记录必为关键字最小的记录，然后依c[i]值的大小对a中记录进行重新排列，试编写算法实现上述排序方法。

【分析】计数排序分两步实现。第一步对每个记录统计（关键字）比它小的记录个数时，注意任意两个记录之间应该只进行一次比较；第二步移动记录时顺记录移动方向扫描，如将a[i]移至位置c[i]+1上，再将a[c[i]+1]移至位置a[c[i]+1]+1上……移动记录的同时还应修改数组c中的相应值。用以下数据检验算法：（28，37，15，40，62，51）。

【解答】算法如下：

```
void Count_Sort(int a[ ]，int n)//计数排序算法
{
  int c[MAXSIZE];
  for(i=0;i<n;i++) //对每一个元素
  {
    for(j=0，count=0;j<n;j++) //统计关键字比它小的元素个数
      if(a[j]<a[i]) count++;
    c[i]=count;
  }
  for(i=0;i<n;i++) //依次求出关键字最小，第二小，…，最大的记录
  {
    min=0;
    for(j=0;j<n;j++)
      if(c[j]<c[min]) min=j; //求出最小记录的下标min
    a[i]<->a[min]; //与第i个记录交换
    c[min]=INFINITY; //修改该记录的c值为无穷大以便下一次选取
  }
}//Count_Sort
```

28．假设含n个记录的序列中，其所有关键字为值介于v和w之间的整数，且其中很多关键字的值是相同的。则可按如下方法进行排序：另设数组number[v..w]且令number[i]统计关键字取整数i的记录数，之后按number重排序列以达到有序。试编写算法实现上述排序方法，并讨论此种方法的优缺点。

【分析】这种排序方法实际上也是一种计数排序。值得注意的是，计数排序和三元组表表示稀疏矩阵时的运算方法有相似之处，在算法中参考了三元组稀疏矩阵转置的算法思想。

【解答】算法如下：

```
void Enum_Sort(int a[ ]，int n)//对关键字只能取v到w之间任意整数的序列进行排序
{
  int number[w+1]，pos[w+1];
  for(i=0;i<n;i++) number[a[i]]++; //计数
  for(pos[0]=0，i=1;i<n;i++)
    pos[i]=pos[i−1]+num[i]; //pos数组可以把关键字的值映射为元素在排好的序列中的位置
  for(i=0;i<n;i++) //构造有序数组c
    c[pos[a[i]]++]=a[i];
  for(i=0;i<n;i++)
    a[i]=c[i];
}//Enum_Sort
```

29．试编写算法，借助“计数”实现基数排序。

【分析】因为计数排序是一种稳定的排序方法，所以，它能够被用来实现基数排序。

【解答】算法如下：

```
typedef enum {0，1，2，3，4，5，6，7，8，9} digit; //个位数类型
typedef digit[3] num; //3位自然数类型，假设低位存储在低下标，高位存储在高下标
void Enum_Radix_Sort(num a[ ]，int n)//利用计数实现基数排序，其中关键字为3位自然数，共有n个
                                    //自然数
{
  int number，pos;
  num c[MAXSIZE];
  for(j=0;j<3;j++) //依次对个位，十位和百位排序
  {
    for(i=0;i<n;i++) number[a[i][j]]++; //计数
    for(pos[0]=0，i=1;i<n;i++)
      pos[i]=pos[i−1]+num[i]; //把关键字的值映射为元素在排好的序列中的位置
    for(i=0;i<n;i++) //构造有序数组c
      c[pos[a[i][j]]++]=a[i];
    for(i=0;i<n;i++)
      a[i]=c[i];
  }//for
}//Enum_Radix_Sort
```

30．序列b的每个元素是一个记录，每个记录占的存储量比其关键字占的存储量大得多，因而记录的移动操作是极为费时的。试写一个算法，将序列b的排序结果放入序列a中，且每个记录只拷贝一次而无其他移动。你的算法可以调用本章中给出的任何排序算法。思考：

当记录存于链表中时，若希望利用快速排序算法对关键字排序，从而最后实现链表的排序，如何模仿上述方法实现？

【分析】可另设一个序列d[1..n]，每个分量含两个域：关键字域存放每个记录的关键字，即d[i].key=b[i].key；地址域指示记录在排序过程中应处的相对位置，其初值应为d[i].pos=i。调用任何一种排序方法调整d[i].pos的值(i=1，2，…，n)。排序结束时表明，序列a中记录应按d[i].pos的值进行排列，即a[i]应调整为b[d[i].pos]。

【解答】算法如下：

```
typedef struct {
                    int key;
                    int pos;
                } Shadow; //影子序列的记录类型
void Shadow_Sort(Rectype b[ ]，Rectype &a[ ]，int n)//对元素很大的记录序列b进行排序，结果放入
                                                    //a中，不移动元素
{
  Shadow d[MAXSIZE];
  for(i=0;i<n;i++) //生成影子序列
  {
    d[i].pos=i;
  }
  for(i=n-1，change=1;i>1&&change;i--) //对影子序列执行冒泡排序
  {
    change=0;
    for(j=0;j<i;j++)
      if(d[j].key>d[j+1].key)
      {
        d[j]<->d[j+1];
        change=1;
      }
  }//for
  for(i=0;i<n;i++) //按照影子序列里记录的原来位置复制原序列
    a[i]=b[d[i].pos];
}//Shadow_Sort
```

10.4 考研真题分析

例题10-1 （清华大学1998年试题）

如果待排序中两个数据元素具有相同的值，在排序前后他们的相互位置发生颠倒，则称该排序算法是不稳定的，_____就是不稳定的排序算法。

A. 起泡排序　B. 归并排序　C. Shell排序　D. 直接插入排序　E. 简单选择排序

【解答】C

例题10-2 （中国科学院计算技术研究所1999年试题）

若有n个元素已构成一个小根堆，那么如果增加一个元素k_{n+1}，请用文字简要说明你如何在log2n的时间内将其重新调整为一个堆？

【解答】将k_{n+1}插入数组的第n+1个位置（即作为一个树叶插入），然后将其与双亲比较，若它大于其双亲则停止调整，否则将k_{n+1}与其父亲交换，重复的将k_{n+1}与新的双亲比较，算法终止于k_{n+1}大于等于其双亲或k_{n+1}本身已上升为根。

例题10-3　（中国科学院软件研究所1999年试题）

已知关键字序列（K_1，K_2，K_3，…，K_{n-1}）是大根堆。

① 试写一算法将（K_1，K_2，K_3，…，K_{n-1}，K_n）调整为大根堆；

② 利用①的算法写一个建大根堆的算法。

【解答】① 将（K_1，K_2，K_3，…，K_{n-1}，K_n）调整为大根堆的算法为：

```
PROCEDURE append(VAR k: ARRAY[1..maxsize] OF keytype;n:integer)
BEGIN
    x:keytype;
    i，n，j:integer;
    x:=k[n];
    i:=[n/2];
    j:=n;
    WHILE (x>k[i]) AND (i>=1) DO
        k[j]:=k[i];
        j:=i;
        i:=[i/2];
    ENDWHILE
    k[j]:=x
END
```

② 利用①的算法建大根堆的算法为：

```
PROCEDURE buildheap(VAR k: filetype;n: integer)
BEGIN
    FOR i:=2 TO n DO
        append(k，i);
    ENDFOR
END
```

例题10-4　（清华大学2000年试题）

有一种简单的排序算法，叫做计数排序（Count Sorting）。这种排序算法对一个待排序的表（用数组表示）进行排序，并将排序结果存放到另一个新的表中。必须注意的是，表中所有待排序的关键码互不相同。计数排序算法针对表中的每个记录，扫描待排序的表一趟，统计表中有多少个记录的关键码比该记录的关键码小。假设针对某一记录，统计出的计数值为c，那么，这个记录在新的有序表中的合适的存放位置即为c。

① 给出适用于计数排序的数据表定义。

② 使用Pascal或C语言编写实现计数排序的算法。

③ 对于有n个记录的表，关键码比较次数是多少？

④ 与简单选择排序相比较，这种方法是否更好？为什么？

【解答】① 数据表定义为：

```
TYPE sortlist=ARRAY [0..n-1] OF key
```

② 用C语言实现的计数排序算法为：

```
void countsort(key*oldlist，key*newlist，int n)
{     int i，j，count;
      for(i:=0; i<n;i++){
            count=0;
            for(j=0;j<n;j++)
                  if(*(oldlist+j)<*(oldlist+i)) count++;
            *(newlist+count)=*(oldlist+i);
      }
}
```

③ 对于有n个记录的表，关键码比较次数是n^2。

④ 简单选择排序比这种排序好，因为对具有n个记录的数据表进行简单选择排序只需进行$1+2+\cdots+(n-1)=\frac{n(n-1)}{2}$次比较，且可在原地进行排序。

例题10-5 （清华大学1998年试题）

如果只想得到1000个元素组成的序列中第5个最小元素之前的部分排序的序列，用____方法最快。

A. 起泡排序　B. 快速排序　C. Shell排序　D. 堆排序　E. 简单选择排序

【分析】简单选择排序第一趟找出最小的元素；第二趟找出第二小的元素；……；只需要5趟就可以得到题目所要求的序列。

【解答】E

例题10-6 （中国科学院软件研究所1998年试题）

若要尽可能快地完成对实数数组的排序，且要求排序是稳定的，则应选_____。

A. 快速排序　B. 堆排序　C. 归并排序　D. 基数排序

【分析】快速排序是不稳定的，排除。

堆排序也是不稳定的，排除。

归并排序稳定，且可以对实数排序，候选。

基数排序不能对实数排序，排除。

【解答】C

例题10-7 （中国科学院计算技术研究所1998年试题）

若需在$O(n\log_2 n)$的时间内完成对数组的排序，且要求排序是稳定的，则可选择的排序方法是_____。

A. 快速排序　　B. 堆排序　　C. 归并排序　　D. 直接插入排序

【分析】快速排序和堆排序都是不稳定的，排除。
归并排序稳定，时间复杂度为$O(n\log_2 n)$，满足条件。
直接插入排序，时间复杂度为$O(n^2)$，排除。
【解答】C

例题10-8 （中国科学院计算技术研究所1998年试题）

已知待排序的n个元素可分为n/k个组，每个组包含k个元素，且任一组内的各元素均分别大于前一组内的所有元素和小于后一组内的所有元素，若采用基于比较的排序，其时间下界应为_____。

A. $O(k\log_2 k)$　　B. $O(k\log_2 n)$　　C. $O(n\log_2 k)$　　D. $O(n\log_2 n)$

【分析】总共分为n/k组，每组有k个元素。由题意，只需对每一组排序即可。
基于比较的排序，长度为k的数组，其时间下界为$O(k\log_2 k)$。
总的时间下界为$n/k*k\log_2 k=n\log_2 k$。
【解答】C

例题10-9 （东北大学1999年试题）

设有1000个无序的元素，希望用最快的速度挑选出其中前10个最大的元素，在以下的排序方法中采用哪一种最好？为什么？

A. 快速排序　B. 归并排序　　C. 堆排序　　D. 基数排序　　E. Shell排序

【解答】采用A最好。可以每次对第一个子序列进行分割，直至子序列长度小于等于10，若长度不足10，则对每两个子序列分割出相应长度的子序列即可。

例题10-10 （清华大学1999年试题）

快速排序的最大递归深度是多少？最小递归深度是多少？

【解答】最大递归深度为n。最小递归深度为$[\log_2 n]+1$。

例题10-11 （北京邮电大学1998年试题）

判断正误：
快速排序的速度在所有排序方法中为最快，而且所需附加空间也最少。

【分析】快速排序需要一个额外的栈空间，比堆排序所需空间要大。
【解答】 错误。

例题10-12 （北京工业大学1998年试题）

2路插入排序是将待排关键字序列r[1..n]中关键字分2路分别按序插入到辅助向量d[1..n]前半部和后半部（注：向量Dsk 视为循环表），其原则为：先将r[1]赋给d[1]，再从r[2]记录开始分2路插入。编写实现二路插入排序算法。

【解答】算法如下：

```
PROCEDURE bin_insert(VAR d，r);
BEGIN
    d[1]:=r[1];
    first:=1;
    last:=1;
    FOR i:=2 TO n DO
        BEGIN
            IF r[i]<r[1] THEN
                IF first=1 THEN
                        BEGIN
                            first:=n;
                            d[n]:=r[i];
                        END;
                    ELSE
                        BEGIN
                            j:=first;
                            WHILE r[i]>d[j] DO d[j−1]:=d[j];
                            d[j]:=r[i];
                            first:=first−1;
                        END;
                    ELSE
                        BEGIN
                            k:=last;
                            WHILE r[i]<d[k] DO d[k+1]:=d[k];
                            d[k+1]:=r[i];
                            last:=last+1;
                        END;
            END;
END.
```

例题10−13 （中国科学院计算机网络信息中心2001年试题）

单项选择题

若关键字是非负整数，则下列排序中平均速度最快的排序是_____；若要求辅助空间为$O(1)$，则平均速度最快的排序是_____；若要求排序是稳定的，且关键字为实数，则平均速度最快的排序是_____。

A．直接插入排序　　B．直接选择排序　　C．Shell排序

D．冒泡排序　　E．快速排序　　F．堆排序

G．归并排序　　H．基数排序

【解答】H，F，G。

例题10−14 （中国科学院计算机网络信息中心2001年试题）

问答题：

用基数排序将取值范围在1到n^3之间的n个整数排序，试给出其时间复杂度。

【解答】因为n^3需要$O(\lg n)$位整数表达，而基数排序的时间复杂度为$O(d(n+rd))$，这里d

为位数，rd为基。故时间复杂度为$O(n\lg n)$，或表达为$O(n\log_2 n)$，这是因为$\lg n=\dfrac{\log_2 n}{\log_2 10}$。

例题10-15　（中国科学院2000年试题）

在含有n个关键字的小根堆（堆顶元素最小）中，关键字是大的记录有可能存储在_____位置上。

A．$\lfloor n/2\rfloor$　　B．$\lfloor n/2\rfloor-1$　　C．1　　D．$\lfloor n/2\rfloor+2$

【解答】D

例题10-16　（中国科学院计算机网络信息中心2000年试题）

若要求排序是稳定的，且关键字为实数，则在下列排序方法中应选_____为宜。

A．直接插入　　B．直接选择　　C．堆　　D．快速排序　E．基数排序

【解答】A

例题10-17　（中国科学院计算机网络信息中心2000年试题）

在快速排序中，要使最坏情况下的空间复杂度为$O(\log_2 n)$则要对快速排序作_____修改。

A．划分元素为三者取中　　B．采用表排序

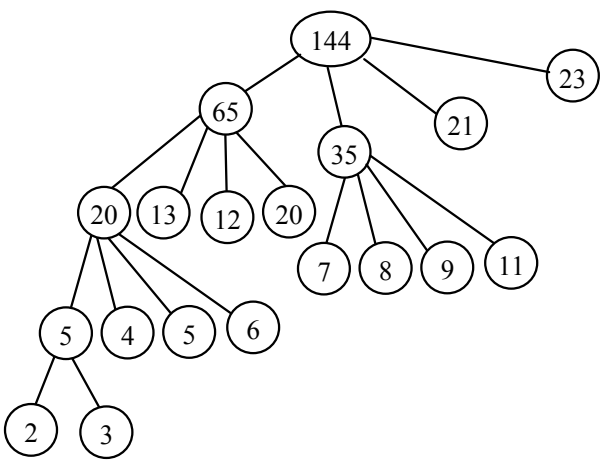
C．先排小集合　　D．先排大集合

【解答】C

例题10-18　（中国科学院计算机网络信息中心2000年试题）

已知在进行置换选择排序时得到14个有序段，其长度分别为 2，3，4，5，6，7，8，9，11，13，17，20，21，23；现进行4-路平衡归并，要求给出所对应的最佳归并树和总的读/写次数。

【解答】



总读写次数：279×2=558

例题10-19（华北计算技术研究所2001年试题）

排序是最常用的算法，请回答下列问题：

（1）请举出5种常见的排序算法，并简单加以说明；

（2）以下列序列{503，087，512，104，908，897，275，503，104，426}为例，给出手工执行上述5种算法后的结果；

（3）请指出上述5种算法中，哪些是稳定的，哪些是不稳定的。

【解答】

（1）5种常见的排序算法。

直接插入排序：将一个记录直接插入到已排好序的有序表中。

希尔排序：分割整个待排序的记录为若干子序列，并将它们分别进行直接插入排序到基本有序，再对全体记录进行直接插入排序。

快速排序：通过一趟排序将排序记录分割成独立的两部分，其中一部分的关键字均比另一部分的关键字小，分别对两个部分记录排序以达到整个记录有序。

归并排序：将两个或两个以上的有序表组合成一个新的有序表。

基数排序：借助多关键字排序的思想对单逻辑关键字进行排序。

（2）手工执行上述5种算法后的结果。

直接插入排序：　087，104，104，275，426，503，503，512，897，908

希尔排序：　087，104，104，275，426，503，503，512，897，908

快速排序：　087，104，104，275，426，503，503，512，897，908

归并排序：　087，104，104，275，426，503，503，512，897，908

基数排序：　087，104，104，275，426，503，503，512，897，908

（3）堆排序和快速排序不稳定，其他均为稳定。

例题10-20 （北京大学2000年试题）

下面PASCAL程序的功能是：用基数排序法对读入的*n*个无符号整数进行排序（排成从小到大次序），在程序空缺处填上适当字句，使其能正确执行。

```
PROGRAM test(input，output);
    CONST M=1000;
          RADIX=16;
          BITS=4;
    TYPE ta=ARRAY[L..M] OF integer;
          tcount=ARRAY[0..RADIX-1] OF integer;
    VAR a:ta; i，n:integer;

    PROCEDURE rsort(VAR a:ta; n:integer);
          VAR t:ta;
                bit，divisor，i:integer;
                count:tcount;
    BEGIN
```

```
            divisor:=1;
            FOR bit:=1 TO BITS DO
            BEGIN
                FOR i:=0 TO RADIX-1 DO
                count[i]:=0;
                FOR i:=1 TO n DO
                    count[a[i] DIV divisor MOD RADIS]:=
                        count[a[i] DIV divisor MOD RADIX]+1;
                FOR i:=1 TO RADIX-1 DO
                count[i]:=count[i]+count[i-1];
                FOR i:=n DOWNTO 1 DO
                BEGIN
                    t[ ____(1)____ ]:=a[i];
                    count[a[i] DIV divisor MOD RADIX]:= ____(2)____;
                END;
                FOR i:=1 TO n DO
                    ____(3)____;
                divisor:=____(4)____;
            END
        END;
BEGIN
        writeln('Input n:');
        readln(n);
        written('Input，'n'，  unsigned integers please:');
        FOR i:=1 TO n DO
            read(a[i]);
        rsort(a，n);
END
```

【解答】

（1）　count[a[i] DIV divisor MOD RADIX]
（2）　count[a[i] DIV divisor MOD RADIX]-1
（3）　d[i]=t[i]
（4）　divisor*RADIX

例题10-21　（中国人民大学2002年试题）

简述如何根据不同的实际情况和要求选择不同的排序算法。

【解答】排序算法主要有简单排序、快速排序、堆排序、归并排序、基数排序。

（1）从平均时间性能而言，快速排序最佳，其所需时间最省，但快速排序在最坏情况下的时间性能不如堆排序和归并排序，在*n*较大时，归并排序所需时间较堆排序省，但它所需的辅助存储量最多。

（2）“简单排序”包括除希尔排序之外的所有插入排序，起泡排序和简单排序，其中直接插入排序为最简单，当序列中的记录“基本有序”或*n*值较小时，它是最佳排序方法，因此常将它和其他的排序方法，诸如快速排序、归并排序等结合在一起使用。

（3）基数排序的时间复杂度也可写成$O(d*n)$。因此，它最适用于*n*值很大而关键字较小的序列。若关键字也很大，而序列中大多数记录的“最高位关键字”均不同，则亦可先

按“最高位关键字”不同将序列分成若干“小”的子序列，而后进行直接插入排序。

（4）从方法的稳定性来比较，基数排序是稳定的内排方法，所有时间复杂度为$O(n^2)$的简单排序法也是稳定的，然而，快速排序、堆排序和希尔排序等时间性能较好的排序方法都是不稳定的。

例题10-22 （北京科技大学2002年试题）

请指出三个稳定的和三个不稳定的内排序方法。

【解答】稳定的排序方法：基数排序、简单排序、插入排序

不稳定的排序方法：快速排序、堆排序、希尔排序

例题10-23 （北京科技大学2002年试题）

设记录的关键字（key）集合K={26，36，41，44，15，68，12，6，51，25}

设以K中第一个关键字（26）为枢轴，写出对K按“快速排序”方法排序时，第一趟排序结束时的结果，并将K调整成一个堆顶元素取最大值的堆。

【解答】

26	36	41	44	15	68	12	6	51	25
6	36	41	44	15	68	12	26	51	25
6	26	41	44	15	68	12	36	51	25
6	12	41	44	15	68	26	36	51	25
6	12	26	44	15	68	41	36	51	25
6	12	15	44	26	68	41	36	51	25
6	12	15	26	44	68	41	36	51	25

例题10-24 （武汉理工大学2002年试题）

选择题：

用冒泡排序的方法对n个记录进行排序，第一趟共要比较__A__对元素。对n个数据进行排序，不稳定排序是__B__，快速排序是一种__C__，关键字序列__D__是一个堆。

供选择的答案：

A：① n-1　② n/2　③ n+1　④ n

B：① 直接插入排序　② 冒泡排序　③ Shell排序　④ 归并排序

C：① 插入排序　② 交换排序　③ 枚举排序　④ 选择排序

D：① 20，76，35，23，80，54　② 20，54，23，80，35，76

③ 80，23，35，76，20，54　④ 20，35，23，80，54，76

【解答】A. ①　B. ③　C. ②　D. ④

例题10-25 （武汉理工大学2002年试题）

判断题：

冒泡排序和堆排序都属于交换排序。

【解答】错误。

例题10-26 （武汉理工大学2002年试题）

算法设计题：

给定关键字k(1<k<n)，要求在无序记录a[1..n]中找到关键字从小到大排序的第k位上的记录，用算法实现。

【解答】

```
ElemType Find (ElemType A[ ]，int n，int k)
{     ElemType x;
      int i，j;
      for(i=1;i<n;i++)
      {
            x=A[i];
            for(j=i-1;j<=0;j--)
                  if(x<A[j]) A[j+1]:=A[j];
                  else break;
                  A[j+1]=x;
      }
      return A[k-1];
}
```

例题10-27 （南京理工大学2002年试题）

简答题：

按锦标赛排序的思想，决出8名运动员之间的名次排列，至少需要多少场比赛（考虑最坏情况）？

【解答】24

10.5　自 测 题

自测题10-1

快速排序在最坏情况下的时间复杂度为_____。

自测题10-2

在下述排序算法中，所需辅助存储量最多的是_____，所需辅助存储量最少的是_____，平均速度最快的是_____。

A. 快速排序　　B. 归并排序　　C. 堆排序

自测题10-3

从供选择的答案中选出适当字句填入数据流程图中A~E处，写在对应栏内：

设顺序文件M有2000个记录，顺序文件N有3000个记录，每个记录中都含有两个独立的

关键字K_1和K_2，文件M已按K_1的上升顺序排序，文件N已按K_2的上升顺序排序。要求如下图所示的数据流程图把文件M和文件N合并成按K_1或K_2的上升顺序排序的文件L，并使整个过程所需时间最短。

A_____，B_____，C_____，D_____，E_____。

供选择的答案：

① 文件N；

② 按K_1上升顺序排序的文件N；

③ 按K_2上升顺序排序的文件M；

④ 记录；

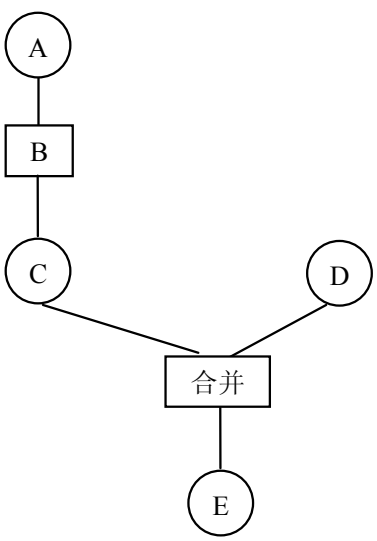
⑤ 按K_1上升的顺序排序；

⑥ 按K_2上升的顺序排序；

⑦ 按K_1上升顺序排序的文件L；

⑧ 按K_2上升顺序排序的文件L；

⑨ 文件M。



自测题10-4

在文件“局部有序”或文件长度较小的情况下，最佳内部排序的方法是_____。

A. 直接插入排序　　B. 起泡排序　　C. 简单选择排序

自测题10-5

快速排序在最坏情况下时间复杂度是$O(n^2)$，比_____的性能差。

A. 堆排序　　B. 起泡排序　　C. 选择排序

自测题10-6

对任意的7个关键字进行排序，至少要进行_____次关键字之间的两两比较。

A. 13　　B. 14　　C. 15　　D. 16　　E. 17

自测题10-7

请编写一过程，对具有*n*个整数的序列进行二叉树排序。

自测题10-8

对具有*n*个元素的序列进行排序时，插入排序法、起泡排序法、堆排序法和二路归并排序法的时间复杂度各是什么？

自测题10-9

若对序列（tang，deng，an，wan，shi，bai，fang，liu）按字典顺序进行排序，在下面的8个序列中，分别指出：

（1）起泡排序第一趟的结果；

（2）初始步长为4的希尔排序第一趟的结果；

（3）以第一个元素为分界元素的快速排序第一趟的结果；

（4）堆排序时的初始堆积。

①（fang，deng，an，liu，shi，bai，tang，wan）

②（an，bai，deng，fang，liu，shi，tang，wan）

③（deng，an，tang，shi，bai，fang，liu，wan）

④（an，deng，tang，wan，shi，bai，fang，liu）

⑤（an，deng，tang，wan，shi，bai，fang，liu）

⑥（wan，tang，fang，liu，shi，bai，an，deng）

⑦（liu，deng，an，fang，shi，bai，tang，wan）

⑧（shi，bai，an，liu，tang，deng，fang，wan）

自测题10-10

设有数组变量说明：

```
VAR a: ARRAY [1..n] OF RECORD key: integer; next: 0..n END;
```

假设各元素的key已有值，并且假设最大值≤9999，再假定在主程序已执行下列语句，已将各元素组成一个链。

```
FOR i:=0 TO n-1 DO a[i].next:=i+1;
a[n]:=0;
```

下面是以10为基的基数排序过程，将数组a按key从小到大排序。

请在划线处填上合适的语句，将程序补充完整。

```
PEOCEDURE bucket_sorting;
    CONST maxkey=9999;
    VAR bucket，  tail: ARRAY [0..9] OF 0..n;
        p，  last: 0..n;
        i: 0..9;
        d: integer;
BEGIN
```

```
        d:=1;
        REPEAT
            FOR i:=0 TO 9 DO tail[i]:=0;
            {分解}
            p:=a[0].next;     {a[0].next是链的第一个结点}
            WHILE p<>0 DO
            BEGIN
                i:=a[p].key DIV d MOD 10;
                IF_____(1)_____
                THEN bucket[i]:=p
                ELSE a[_____(2)_____].next:=p;
                tail[i]:=p;
                _____(3)_____
            END;
            {收集}
              last:=0;
              FOR i:=0 TO 9 DO
                  IF_____(4)_____;
                  THEN BEGIN
                       a[last].next:=_____(5)_____;
                       last:=_____(6)_____
                       END;
              a[last].next:=0;
              d:=_____(7)_____;
        UNTIL d>maxkey
    END {bucket_sorting}
```

自测题10-11

下面是一个起泡排序算法，待排序的线性表存在数组中，结点类型定义如下：

```
node=RECORD
      key: integer;
      info: datatype
END
```

其中key为排序码。下面的算法中*r*是存放待排序线性表的数组，*n*为线性表中结点的个数。

```
1    PROCEDURE bubble_sort(VAR r: ARRAY [1..max] OF node; n: integer);
2          VAR temp: node;
3              i，j: integer;
4              ok: boolean;
5    BEGIN
6          ok:=false;
7        i:=1;
8        WHILE(i<=n-1) AND NOT ok DO
9        BEGIN
10             ok:=true;
11             FOR j:=1 TO n-i DO
12             IF r[j].key >r[j+1].key THEN
13             BEGIN
```

```
14                    ok:=false;
15                    temp:=r[j];
16                    r[j]:=r[j+1];
17                    r[j+1]:=temp
18                END;
19              i:=i+1;
20            END
21        END
```

问题：

（1）如果n=9，r中的排序码值是{38，7，24，14，36，2，95，44，48}写出执行一次WHILE循环后，r中新的排序码值。

（2）该算法是否稳定。

（3）算法中变量ok的作用是什么。

（4）为减少记录交换次数，修改上面的算法，使得在每趟扫描中避免出现连续的记录交换。做法是增加局部整型变量k，并将第11~18行替换成以下程序。请把其中{? i}处替换为适当内容，并问修改后的算法是否稳定。

```
j:=1;
WHILE j<=n-i DO
BEGIN
      k:=j+1;
      WHILE (k<=n-i+1) AND {? 1} DO
            k:=k+1;
            IF {? 2} THEN
            BEGIN
                  ok:=false;
                  temp:=r[j];
                  r[j]:=r[k-1];
                  r[k-1]:=temp
            END;
      {? 3}
END;
```


自测题10-12

回答问题并写出推导过程。

对50个整数进行快排需进行关键字间比较次数可能达到的最大值和最小值各为多少？（快排：指快速排序。）

自测题10-13

填空：

（1）若待排序的记录总数为n，则快速排序的总执行时间为_____。

（2）比较次数达到树形选择排序水平，同时又不增加存储开销的排序方法是_____。

自测题10-14

我们知道，待排序记录序列的初始状态会影响某些排序算法的时间量级（包括记录的比较次数和移动次数两个方面）。请在下表的空格内，填上“是”或“否”，以表明使用相应的排序算法时记录的比较次数和移动次数之量级是否受影响，即当待排序记录序列处于某种初始排列状态时排序时间是一个量级；当处于另一种初始排列状态时排序时间又变成另一个量级。另外，在每行的最后一个空格中亦请标明相应的算法是否稳定的排序算法。

	记录的比较次数受影响与否？	记录的移动次数受影响与否？	该排序算法是否稳定？
直接插入排序 二分插入排序 直接选择排序 快速排序 归并排序			

10.6 自测题答案

【自测题10-1】

$O(n^2)$。

【自测题10-2】

在各种排序方法中，快速排序是平均速度最快的排序方法，但因为快速排序是一个递归的过程，在排序过程中需要用到栈。堆排序只需用到一个记录大小的辅助空间。实现归并排序需和待排记录等数量的辅助空间。

答案依次为B，C，A。

【自测题10-3】

答案为A：⑨ B：⑥ C：③ D：① E：⑧

【自测题10-4】

答案为A。

【自测题10-5】

答案为A。

【自测题10-6】

答案为A。

【自测题10-7】

```
struct bintree{
        int ele;
        struct bintree *left;
        struct bintree *right;
```

```
}

binsort(a，n，p_tree)
int *a;     /*a[n]*/
int n;
struct bintree *p_tree;         /*p_tree[n]*/
{
        int i;
        boolean greater;
        float value;
        struct bintree *p_ptreel，   *p_treesave;

        for (i=0;i<n;i++){
                p_tree[i].ele=a[i];
                p_tree[i].left=NULL;
                p_tree[i].right=NULL;
        }
        for(i=1;i<n;i++){
                value=a[i];
                p_treel=p_tree;        /*root of bintree*/

                while (p_tree1!=NULL){
                        if (greater=(p_tree->ele>value))
                        P_tree1=p_tree->left;
                        else p_tree1=p_tree->right;
                }
                if(greater)
                        p_treesave->left=p_tree+i;          /*chained onto the tree*/
                else p_treesave->right=p_tree+i;
        }
}
```

【自测题10-8】

插入排序法：$O(n^2)$;

起泡排序法：$O(n^2)$;

堆排序法：$O(n\log n)$;

2路归并排序法：$O(n\log n)$。

【自测题10-9】

（1）起泡排序第一趟的结果；③

（2）初始步长为4的希尔排序第一趟的结果；⑧

（3）以第一个元素为分界元素的快速排序第一趟的结果；⑦

（4）堆排序时的初始堆：②

【自测题10-10】

（1）tail[i]=0

（2）tail[i]

（3）p:=a[p].next

（4）tail[i]<>0
（5）bucket[i]
（6）tail[i]
（7）d*10

【自测题10-11】
（1）一次WHILE循环后，r中新的排序码值为：
{7，24，14，36，2，38，44，48，95}
（2）该算法稳定。
（3）变量*ok*表示是否全部有序。
（4）1: r[j].key>=r[k].key
2: j<>k-1
3: j:=k

修改后的算法不稳定。

【自测题10-12】

最大比较次数：在关键字有序的情况下，比较次数最大。

$$49+48+\cdots+1=1225$$

最小比较次数：若每次都分割成长度相等或长度相差为1的子序列，则比较次数最小。

$$50*5+18*2=286$$

【自测题10-13】

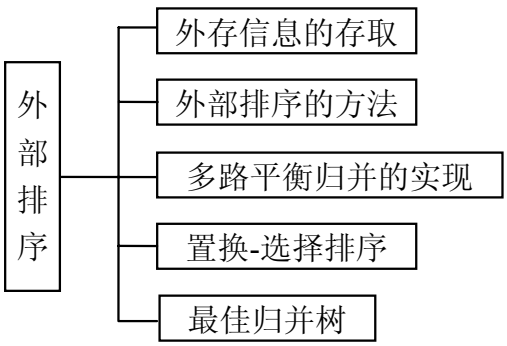
（1）快速排序的总执行时间为$O(n\log n)$。
（2）堆排序方法。

【自测题10-14】

	记录的比较次数受影响与否？	记录的移动次数受影响与否？	该排序算法是否稳定？
直接插入排序	是	是	是
二分插入排序	否	是	否
直接选择排序	否	是	是
快速排序	是	是	否
归并排序	否	否	是

第 11 章　外 部 排 序

11.1　基本知识结构图



11.2　知 识 点

1. 外部排序的方法

外部排序指的是待排序记录的数量很大，以致内存一次不能容纳全部记录，在排序过程中尚需对外存进行访问的排序过程。

外部排序基本上由两个相对独立的阶段组成。首先，按可用内存大小，将外存上含n个记录的文件分成若干长度为l的子文件或段，依次读入内存并利用有效的内部排序方法对它们进行排序，并将排序后得到的有序子文件（即归并段或顺串）重新写入外存；然后，对这些归并段进行逐趟归并，使归并段（有序的子文件）逐渐由小至大，直至得到整个有序文件为止。

2. 多路平衡归并及各种优化策略

可以利用“败者树”进行k-路平衡归并，使在k个记录中选出关键字最小的记录时需进行的比较次数减少，从而使总的归并时间降低。

置换-选择排序是通过减少外部文件经过内部排序之后得到的初始归并段的个数来减少归并趟数，从而提高外部排序效率的方法。它是在树形选择排序的基础上得来的，它的特点是：在整个排序（得到所有初始归并段）的过程中，选择最小（或最大）关键字和输入、输出交叉或平行进行。

若对通过置换-选择排序获得的长度不等的m个初始归并段，构造一棵哈夫曼树作为归并树，便可使在进行外部归并时所需对外存进行的读/写次数达最少，这棵归并树就称作最佳归并树。

11.3 习题及参考答案

1．假设某文件经内部排序得到100个初始归并段，试问：

（1）若要使多路归并三趟完成排序，则应取归并的路数至少为多少？

（2）假若操作系统要求一个程序同时可用的输入、输出文件的总数不超过13，则按多路归并至少需几趟可完成排序？如果限定这个趟数，则可取的最低路数是多少？

【解答】（1）至少取5-路进行归并；

（2）每次可取12路进行归并，则至少需2趟完成排序。然而对总数为100的初始归并段，要在2趟内完成归并，进行10-路归并即可。

2．假设一次I/O的物理块大小为150，每次可对750个记录进行内部排序，那么，对含有150 000个记录的磁盘文件进行4-路平衡归并排序时，需进行多少次I/O？

【解答】150 000个记录（1 000个物理块）经内部排序后得到200个初始归并段，需进行$[\log_4 200]=4$趟4-路平衡归并排序。所以，总的I/O次数为：$1000\times2+1000\times2\times4=10000$。

3．“败者树”中的“败者”指的是什么？若利用败者树求k个数中的最大值，在某次比较中得到$a>b$，那么谁是败者？“败者树”与“堆”有何区别？

【解答】败者树中的败者是在刚刚进行的比较中失败了的，记录在双亲结点中。b是败者。败者树与堆的最大差别在于：败者树是由参加比较的n个元素作为叶子结点而得到的完全二叉树，而“堆”则是n个元素$R_i(i=1，2，\cdots，n)$的序列，它满足下列性质：$R_i\leqslant R_{2i}$且$R_i\leqslant R_{2i+1}$ $(1\leqslant i\leqslant[n/2])$。由于这个性质中下标$i$和$2i$，$2i+1$的关系恰好和完全二叉树中第$i$个结点和它的孩子结点的序号之间的关系一致，则堆可看成是含n个结点的完全二叉树。

11.4 考研真题分析

例题11-1 （中国科学院软件研究所1998年试题）

对外部排序的K路平衡归并，采用败者树时，归并效率与K_____。

A. 有关　　B. 无关

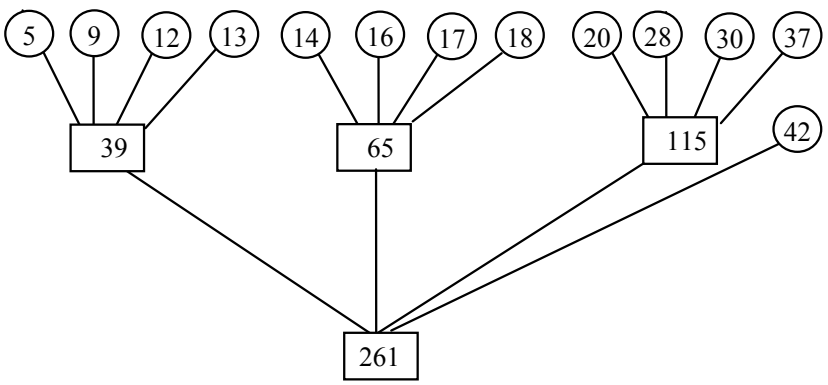
【解答】B

例题11-2（清华大学1998年试题）

设有13个初始归并段，其长度分别为28、16、37、42、5、9、13、14、20、17、30、12和18。试画出4路归并时的最佳归并树，并计算它的带权路径长度WPL。

【分析】在一般情况下，对k路归并而言，若有m个初始归并段，若$(m-1)\text{MOD}(k-1)=0$，则不需要加虚段，否则需附加$k-(m-1)(k-1)-1$个虚段，此时，每一次归并为$(m-1)\text{MOD}(k-1)+1$路归并。

【解答】最佳归并树如下图所示。



WPL=(5+9+12+13+14+16+17+18+20+28+30+37)×2+42=480

例题11-3　（清华大学1998年试题）

设有11个长度（即包含记录个数）不同的初始归并段，它们所包含的记录个数分别为25、40、16、38、77、64、53、88、9、48和98。

试根据它们做4路平衡归并，要求：

① 指出总的归并趟数；

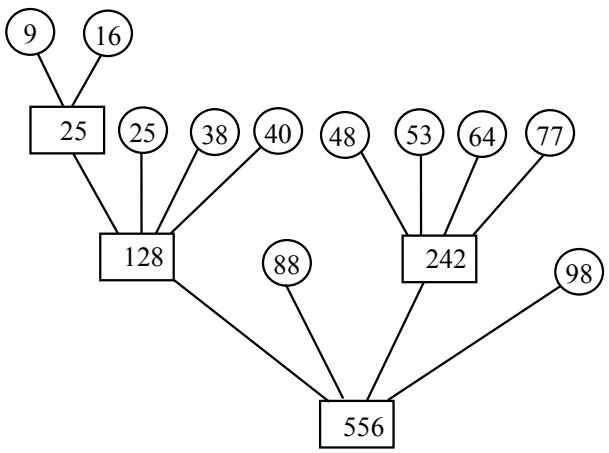
② 构造最佳归并树；

③ 根据最佳归并树计算每一趟及总的读记录数。

【解答】

① 4路平衡归并总的归并趟数$[\log_4 11]=2$。

② 构造最佳归并树如下图所示。



③ 根据最佳归并树计算每一趟及总的读记录数。

第一趟的读记录数：9+16=25

第二趟的读记录数：25+25+38+40+48+53+64+77=128+242=370
第三趟的读记录数：128+88+242+98=556
总的读记录数：25+370+556=951

例题11-4（北京邮电大学1998年试题）

文件中记录的关键字分别为41、39、28、32、22、19、11、50、13、21、1、33、37、3、52、16、4、8、72、12和32。设缓冲区W有能容纳5个记录的容量。按置换选择排序的方法求初始归并段。请定出各初始归并段的关键字。

【分析】求初始归并段的算法如下：

假设初始待排序文件为输入文件FI，初始归并段文件为输出文件FO，内存工作区为WA，FO和WA的初始状态为空，并设内存工作区WA可容纳w个记录。操作如下：

（1）从FI输入w个记录到工作区WA。
（2）从WA中选其中关键字取最小值的记录，记为MINMAX记录。
（3）将MINMAX记录输出到FO中去。
（4）若FI不空，则从FI输入下一个记录到WA中。
（5）从WA中所有关键字比MINMAX记录的关键字大的记录中选出最小关键字记录，作为新的MINMAX记录。
（6）重复（3）～（5），直至在WA中选不出新的MINMAX为止，由此得到一个初始归并段，输出一个归并段的结束标志到FO中去。
（7）重复（2）~（6），直至WA为空，由此得到全部初始归并段。

【解答】各初始归并段如下：

RUN1：22，28，32，39，41，50　　关键字：50
RUN2：1，11，13，19，21，33，37，52，72　　关键字：72
RUN3：3，4，8，12，16，32　　关键字：32

例题11-5 （上海交通大学1999年试题）

给出一组关键字T=（12，2，16，30，8，28，4，10，20，6，18），设内存工作区可容纳4个记录，写出用置换-选择排序得到的全部初始归并段。

【解答】RUN1：2，8，12，16，28，30。
RUN2：4，6，10，18，20。

11.5 自 测 题

自测题11-1

在进行磁带文件分类时，通常用置换-选择分类方法生成最初合并段。若内存缓冲区只能存放m个记录，则平均情况下的最初合并段的长度是多少？并请进行简单的分析。有无

可能只生成一个最初合并段，即该合并段的记录总数和被分类的磁带文件记录总数相等？为什么？

自测题11-2

判断正误：

减少初始归并段的数量，可使得外排序的时间缩短。

自测题11-3

判断正误：

① 一般说来，外排序所需要的总时间=内排序时间+外存信息读写时间+内部归并所需要的时间。

② 采用多路归并方法可以减少初始归并段的数量。

③ 外排序过程主要分为两个阶段：生成初始归并阶段和对归并段进行再归并的阶段。

自测题11-4

归并段长度分别为9、4、7、3、8、6和15。试画出3路平衡最佳归并树。

自测题11-5

已知有31个长度不等的初始归并段，其中8段长度为2，8段长度为3，7段长度为5，5段长度为12，3段长度为20（单位均为物理块），请为此设计一个最佳5-路归并方案，并计算总的（归并所需的）读/写外存的次数。

11.6　自测题答案

【自测题11-1】

平均情况下，最初合并段的长度为$2m$。

当记录已经全部有序时，可能生成一个最初合并段。

【自测题11-2】

正确。

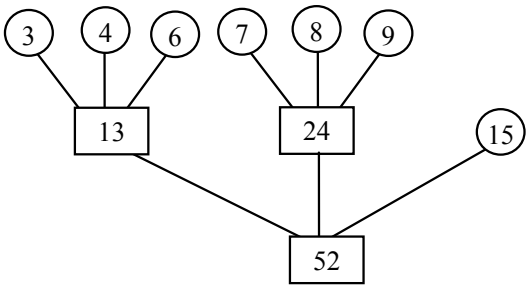
【自测题11-3】

（1）正确。

（2）错误。

（3）正确。

【自测题11-4】所求的3-路平衡最佳归并树如下图所示。



【自测题11-5】

最佳5-路归并树如下图所示。

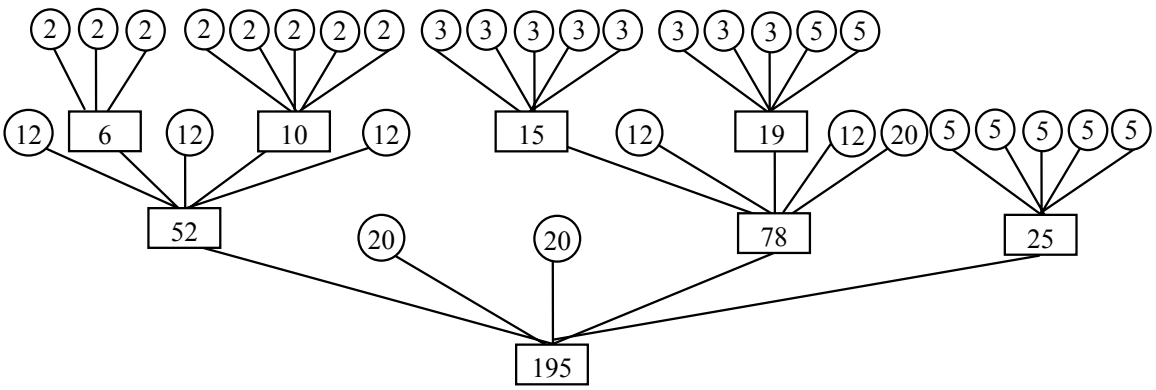
读写记录数

第一趟：2*8+3*8+5*2=50

第二趟：6+10+12*5+15+19+20+5*5=155

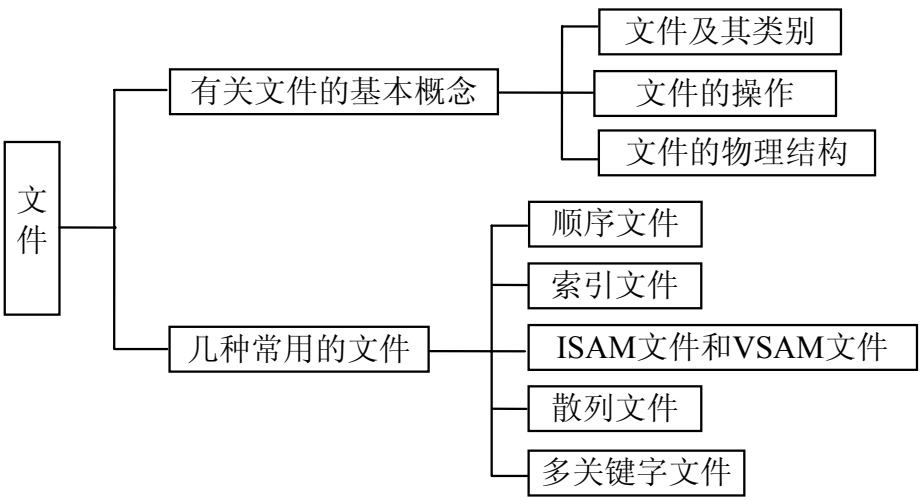
第三趟：20+52+20+78+25=195

总读写记录数：50+155+195=400



第 12 章　文　件

12.1　基本知识结构图



12.2　知 识 点

1. 有关文件的基本概念

文件是由大量性质相同的记录组成的集合。可按记录的类型不同而分成两类，即操作系统的文件和数据库文件；还可按记录的特性分成定长记录文件和不定长记录文件。可对文件进行检索和修改操作。检索有三种方式，即顺序存取、直接存取和按关键字存取。修改包括插入一个记录、删除一个记录和更新一个记录3种操作。

2. 几种常用的文件的物理结构

文件在存储介质（磁盘或磁带）上的组织方式称为文件的物理结构。常用的存储方式有顺序文件、索引文件、索引顺序存取方法（ISAM）文件、虚拟存储存取方法（VSAM）文件、直接存取文件（散列文件）和多关键字文件。

12.3　考研真题分析

例题12-1 （中国科学院软件研究所1998年试题）

用ISAM组织文件适合于_____。

A. 磁带　　B. 磁盘

【分析】索引顺序存取方法ISAM为Indexed Sequential Access Method的缩写，是一种专为磁盘存取设计的文件组织形式。

【解答】B

例题12-2（中国科学院计算技术研究所1998年试题）

用ISAM和VSAM组织文件属于_____。

A. 顺序文件　　B. 索引文件　　C. 散列文件

【分析】ISAM（Indexed Sequential Access Method）：索引顺序存取方法。

VSAM（Virtual Storage Access Method）：虚拟存储存取方法。

【解答】B

例题12-3 （中国科学院自动化研究所1998年试题）

试解释下列名词：

顺序文件、散列文件、索引文件、倒排文件。

【解答】顺序文件：记录按其在文件中的逻辑顺序依次进入存储介质而建立的文件。在顺序文件中，物理记录的顺序和逻辑记录的顺序是一致的。

散列文件：利用杂凑（hash）法进行组织的文件。

索引文件：包括文件数据区和索引表两大部分的文件称为索引文件。

倒排文件：次关键字索引为倒排表的多关键字文件称为倒排文件。

12.4 自 测 题

自测题12-1

判断正误：

检索出文件中关键码值落在某个连续范围内的全部记录，这种操作称为范围检索。对经常需要作范围检索的文件进行组织，采用散列法优于采用顺序索引法。

自测题12-2

已知职工文件中包括职工号、职工姓名、职务和职称4个数据项（见下表）。职务有校长、系主任、室主任和教员；校长领导所有系主任，系主任领导他所在系的所有室主任；室主任领导他所在室的全体教员；职称有教授、副教授和讲师3种。请在职工文件的数据结构上设置若干指针项和索引，以满足下列两种查找的需要：

（1）能够检索出全体职工间领导与被领导的情况；

（2）能够分别检索出全体教授、全体副教授、全体讲师。

要求指针数量尽可能少，给出各指针项索引的名称及含义即可。

职工文件表

职工号	职工姓名	职务	职称
001	张军	教员	讲师
002	沈灵	系主任	教授
003	叶明	校长	教授
004	张莲	室主任	副教授
005	叶宏	系主任	教授
006	周芳	教员	教授
007	刘光	系主任	教授
008	黄兵	教员	讲师
009	李民	室主任	教授
010	赵松	教员	副教授
…	…	…	…

12.5 自测题答案

【自测题12-1】

正确。

【自测题12-2】

在职务项中增加一个指针项，指向其领导者。

在职称项中增加一个指针项，指向同一职称的下一个职工。

增加一个索引表：

关键字	头指针	长度
讲师	001	
副教授	004	
教授	002	

第 13 章　名 校 试 题

哈尔滨工业大学2002年硕士研究生入学考试试题

一、填空（每空1-2分，共20分）

1．具有n个顶点的开放树，边的总数有_____条。
2．由三个结点组成的二叉树共有_____种不同的结构形态。（2分）
3．n个元素的线性表，采用顺序存储结构，插入一个元素要平均移动表中_____个元素，删除一个元素最坏情况下要移动_____个元素。（4分）
4．一个二叉树第5层结点最多有_____个。
5．若在一个表中共有625个元素，且查找每个元素的概率相同，那么在采用分块查找时，每块的最佳长度为_____，此时平均查找长度为_____。（4分）
6．具有n个叶子结点的哈夫曼（Huffman）树中，其结点总数为_____。
7．在折半查找中，要求被查找元素必须采用_____存储结构，且_____。
8．堆排序（heap sort）的时间复杂度为_____。基数排序的时间复杂度为_____。
9．一个无向图有n个顶点，e条边，则所有顶点的度数之和为_____。
10．设F是一个森林，B是由F按自然对应关系转换而得到的二叉树，F中有n个非终端结点，则B中右子树为空的结点有_____个。（2分）

二、单项选择（每空1分，共8分）

1．下三角矩阵$A_{(n\times n)}$按行优先顺序压缩在数组Sa[(n+1)*n/2]中，若非零元素$a_{ij}(0\leqslant i, j<n)$存放在Sa[k]中，则$i$，$j$和$k$之间的关系为（　）。

A．$k=i*n+j$　　B．$k=j*n/2+i$
C．$k=(i+1)*i/2+j$　　D．$k=(j-1)*n/2+i-1$

2．将一株有100个结点的完全二叉树从上到下，从左到右依次进行编号，根结点的编号为1，则编号为49的结点的右孩子编号为（　）。

A．98　　B．99
C．50　　D．没有右孩子

3．数据在计算机中存储器内表示时，物理地址和逻辑地址相同并且是连续的，称之为（　）。

A．逻辑结构　　B．顺序存储结构
C．链式存储结构　　D．以上都对

4．一株非空二叉树的所有叶子结点在先根、中根和后根遍历序列中的相对顺序（ ）。

A．发生改变　　B．不发生改变
C．不能确定　　D．以上都不对

5．对于一个具有n个顶点e条边的无向图，若采用邻接表表示，则表向量（顶上表）的大小为（ ）。所有关于顶点的邻接表（边表）的结点总数为（ ）。

A．n　　B．n+1　　C．n−1　　D．n+e
E．e/2　　F．e　　G．2e　　H．n+e

6．数据在计算机存储器内表示时，根据结点关键字直接计算出该结点的存储地址，这种方法称为（ ）

A．索引存储方式　　B．顺序存储方式
C．链式存储方式　　D．散列存储方式

7. 多关键字文件是指（ ）。

A．有多个主关键字　　B．有一个主关键字和多个次关键字
C．有多个次关键字　　D．有多个主关键字和多个次关键字

三、判断题：正确的打 √，错误的打×（每题1分，共10分）

1．对于任意一个图，从它的某个顶点进行一次先深或先广搜索可以访问到该图的每个顶点。（ ）
2．一个带权的无向连通图的最小生成树的权值之和是惟一的。（ ）
3．树的数组表示法（单链或父链表示法）中兄弟结点的编号不一定是连续的。（ ）
4．二元查找树（binary searching tree）的任何结点的左右子树都是二元查找树。（ ）
5．任何一个AOE（Activity On Edge）网中至少有一条关键路径，且是从源点到汇点的最短的一条路径。（ ）
6．索引顺序文件既可以随机访问，也可以顺序访问。（ ）
7．带头结点的单循环链表中，任一结点的后继结点的指针域均不空。（ ）
8．中根遍历二元查找树（binary searching tree）所得序列一定是有序序列。（ ）
9．栈和队列均为操作受限的线性表。（ ）
10．求有向图任意两点之间的最短路径的Flody算法适合于所有的带权有向图。（ ）

四、简答题（共11分）

1．已知树的广义表表示如下T=(A(B(E(K，L))，C(G)，D(H(M)，I，J)))，画出该广义表所对应的树。（3分）
2．何为稳定分类？举例说明为什么快速分类（Quicksort）是不稳定分类？（5分）
3．何为拓扑排序（topological sort）？（3分）

五、已知如图1所示的二叉树是由某森林转换而来，画出其原来的森林。（4分）

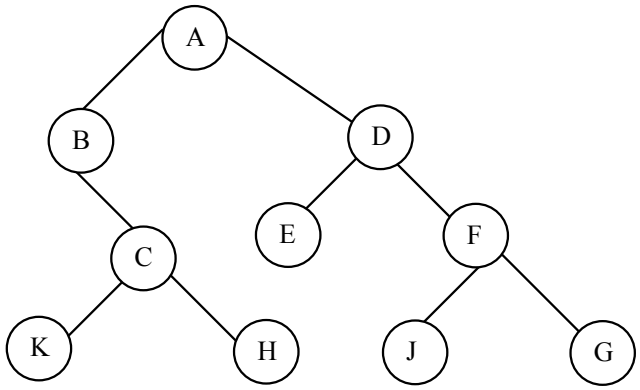


图 1

六、已知带权无向图如图2所示，利用克鲁斯科尔（Kruskal）算法，画出该无向最小生成树的每一步。（4分）

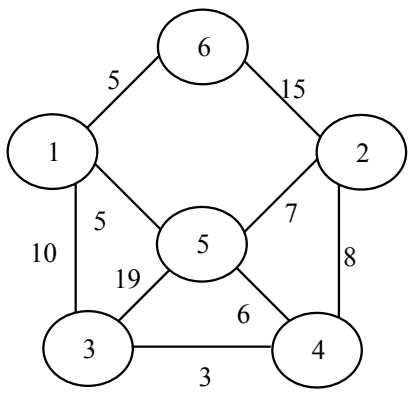


图 2

七、试写出把图的邻接矩阵表示转换为邻接表表示的算法。（8分）

八、设计一算法分别求出二叉树的叶子结点，度数为1的结点，度数为2的结点的个数。（8分）

九、已知无向图$G=(V, E)$，给出求图G的连通分量个数的算法。（9分）

十、设计将数组A[n]中所有的偶数移到奇数之前的算法。要求增加存储空间，且时间复杂度为$O(n)$。（8分）

十一、设有m个连续单元供一个栈与队列使用，且栈与队列的实际占用单元数事先并不知道，但是要求在任何时刻它们占用的单元数量不超过m，试写出上述栈与队列的插入算法。（10分）

参 考 答 案

一、

1．$n-1$　　2．5　　3．$\frac{n}{2}$　$n-1$　　4．16

5．25　26　　6．$2n-1$　　7．顺序　是有序的　　8．$O(n\log n)$　$O(d(n+rd))$

9．$2e$　　10．$n+1$

二、

1．C　　2．B　　3．B　　4．B

5．A，C　　6．D　　7．C

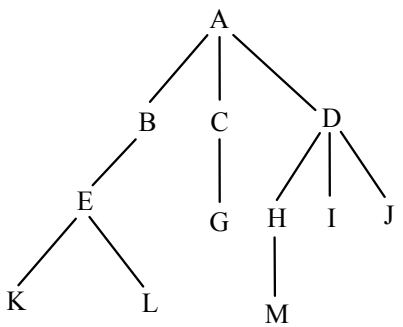
三、

1．×　　2．√　　3．×　　4．√　　5．√

6．×　　7．√　　8．√　　9．√　　10．√

四、

1．



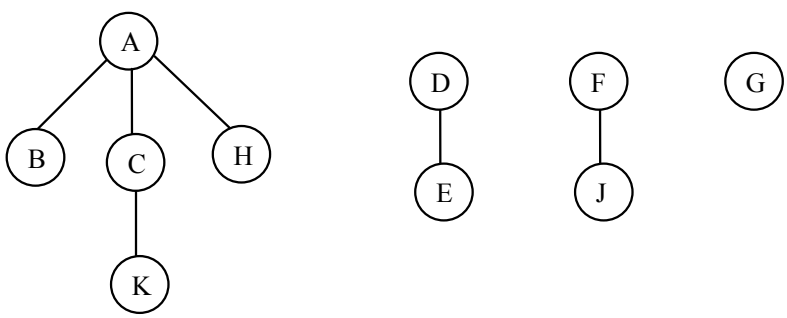
2．对于具有同一排序码的多个记录来说，若采用的排序方法使排序后记录的相对次序不变，则称此排序方法是稳定的，否则称为不稳定的。

例：12，11，18，15，11，经过快速排序后得到的序列为11，11，12，15，18

其中11与11的次序与原记录的次序不相同，所以快速排序是不稳定的排序。

3．由集合上一偏序得到该集合的一个全序的操作称之为拓扑排序。

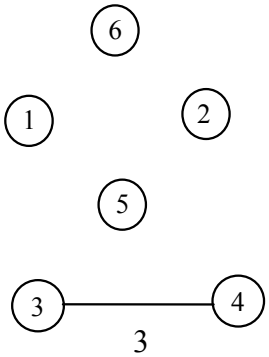
五、



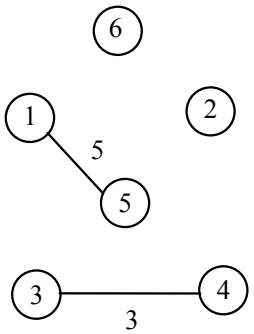
六、

该无向最小生成树可分为五步：

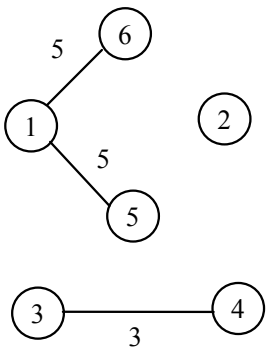
（1）



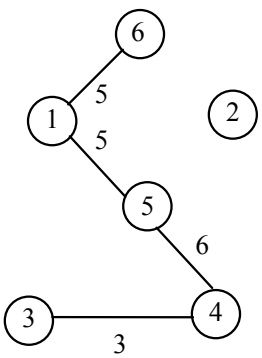
（2）



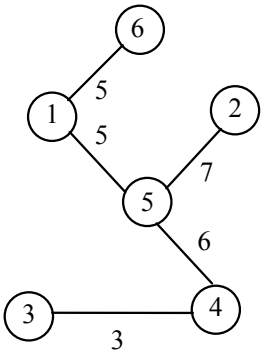
（3）



（4）



（5）



七、

设图的邻接矩阵为G[n][n]（针对无向图的）

定义邻接表结点的类型为：
```
struct edgenode{
        int adjvex;
        edgenode * next;
        }
typedef edgenode * adjlist[n];          //定义adjlist为存储n个表头指针的数组类型
```

将邻接矩阵表示转换为邻接表表示的算法如下：

```
void matrixtolist (int G[ ][ ]，  adjlist GL，  int n)
{
    edgenode *p，*q;
    for (int i=0;i<n;i++) GL[i]=Null;
    for (int i=0;i<n;i++)
        for (int j=0;j<n;j++){
            if (G[i][j]!=0) p=(edgenode *) malloc (sizeof(edgenode));
            p->adjvex=j;
            p->next=Null;
            if (GL[i]== Null){GL[i]=p;q=p;}
            else {q->next=p;
                  q=p;
            }
        }
}
```

八、

定义树的结点类型为：
```
typedef struct node {
        ElemType    data;
        struct    node    *lchild;
        struct    node    *rchild;
        }BTree;
```

定义静态全局变量a[3]，其中a[0]、a[1]、a[2]分别用来存储度数分别为0、1、2结点的个数：

```
static int a[3]
```

算法如下：

```
void countnode (BTree *b)
{
    if (b==Null) return;
    if (b->lchild==Null && b->rchild==Null) a[0]=a[0]+1;
    if (b->lchild!=Null && b->rchild!=Null) a[2]=a[2]+1;
    else a[1]=a[1]+1;
    countnode (b->lchild);
    countnode (b->rchild);
}
```

九、（9分）

以邻接表的存储结构为例，邻接表的数据类型如下：

```
typedef struct {                        //结点类型
    int adjvex;                         //该弧所指向的顶点的位置
    struct ArcNode * nextarc;           //指向下一条弧的指针
    InfoType info;                      //该弧的相关信息
}ArcNode;        N
typedef struct Vnode {                  //表头结点
    Vertex data                         //顶点信息
    ArcNode *firstarc                   //指向第一条弧
    } VNode;
typedef VNode AdjList [maxVertexNum];     // AdjList是邻接表类型
typedef struct {
    AdjList adjlist;                //邻接表
    int n，  e;                     //图中顶点数n和边数e
}ALGraph;                           //图类型
```

算法如下：

```
static int visit [n];
void dfs (ALGraph *g，  int v);
int dfscount (ALGraph *g)
{
    int i，  j; j=0;
    for (i=0; i<g->n; i++)
        if (visit[i]==0){
            j++;
            dfs (g，  i);
        }
}
void dfs (ALGraph *g，  int v)
{
    ArcNode *p;
    visited[v]=1;
    p=g->adjlist[v]，firstarc;
    while (p!=Null) {
        if (visited[p->adjvext]==0)
        dfs (g，p->adjvex);
        p=p->nextarc;
```

```
    }
}
```

十、（8分）

```
void transfer (int a[ ],  int n)
{
    int b[n];
    int k=0;
    int j=n−1;
    for (int i=0; i<n; i++) {
        if (a[i]%2==0) {
            a[k]=a[i];
            k++;
        }else {a[j]=a[i];
            j−−;
        }
        for (int i=0; i<n; i++)a[i]=b[i]
    }
```

十一、（10分）

定义结点的结构为：
```
struct Node {
    ElemType Data;
    struct node *next;
    }
```
定义栈的结构为：
```
struct Stuck {
    Node * base;
    Node * top;
    }
```
定义队列的结构为：
```
struct Queue {
    Node * front;
    Node * tail;
    }
```

设*m*个连续单元的数组为b[m]，定义全局数组static int a[m]用以标识*m*个单元中各个单元是否已被占用：a[i]=1表示已占用；a[i]=0表示未被占用。

```
void insertstack (struct stack&S,  ElemType elemtype)
{
    for (int i=0;i<m;i++)
        if (a[i]==0) break;
    if (i==m) {
        count<<"There is no space"<<end;
        return;
    }
    a[i]=1;
    Node *p;
    P=&b[i];
    p−>data=elemtype;p−>next=Null;
    if (S.BASE==Null) {
            S.base=p;
```

```
                S.top=p;
            }
        else
        {
            p->next=top;
            S.top=p;
        }
}
void insert Queue (struct Queue &Q，  ElemType elemtype)
{
        for (int i=0;i<m;i++)
            if (a[i]==0)break;
        if (i==m){
            count<<"There is no space"<<end;
            return;
        }
        a[i]=1;
        Node *p;
        p=&b[i];
        p->data=elemtype; p->next=Null;
        if (Q.front==Null) {Q.front=p;
                Q.top=p;
        }else {
                Q.top->next=p;
                Q.top=p;
            }
}
```

华北计算技术研究所2002年硕士研究生入学考试试题

一、选择合适的答案，填空回答下列问题。（每空2分，计10分）

1. 设有一个顺序栈S，元素s_1，s_2，s_3，s_4，s_5，s_6依次进栈，如果6个元素的出栈顺序为s_2，s_3，s_4，s_6，s_5，s_1，则顺序栈的容量至少应为_____？
2. 假定在一棵二叉树中，度为2的结点有50个，度为1的结点有20个，度为0的结点有_____个，该二叉树共有_____条边？
3. 有$n(n\geq 1)$个顶点的有向强连通图最少有_____条边。
4. 判断有向图是否存在回路，除了可以利用拓扑排序方法外，还可以利用_____。

二、判断下列叙述的对错。如果正确，在题前打“√”，否则打“×”。（每题2分，计10分）

1. 线性表的顺序存储表示优于链式存储表示。
2. 若有一个叶子结点是二叉树中某个子树的前序遍历结果序列的最后一个结点，则它一定是该子树的中序遍历结果序列的最后一个结点。

3. 在向二叉搜索树中插入新结点时，新结点必须作为叶子结点插入。
4. 在散列法中采取链地址法来解决冲突时，其装填因子的取值一定在（0，1）之间。
5. 静态索引结构是指系统运行时文件的索引结构不会发生改变，而文件的内容是可以改变的。

三、已知有向图的邻接矩阵如下所示，该有向图的顶点分别为A、B、C、D、E、F、G、H、I。（10分）

1. 根据此邻接矩阵画出该图。
2. 给出相应的邻接表。
3. 对该图进行拓扑排序。

有向图的邻接矩阵为

	A	B	C	D	E	F	G	H	I
A	0	1	1	1	0	0	0	0	0
B	0	0	0	0	1	0	0	0	0
C	0	0	0	0	1	1	0	0	0
D	0	0	0	0	0	1	0	0	0
E	0	0	0	0	0	0	1	1	0
F	0	0	0	0	0	0	0	0	1
G	0	0	0	0	0	0	0	0	1
H	0	0	0	0	0	0	0	0	0
I	0	0	0	0	0	0	0	0	0

四、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。使用等长编码表示电文是另一种编码方案。比较两种方案的优缺点。（10分）

五、已知如下所示长度为9的表{16、3、7、11、9、26、18、14、15}（20分）

1. 按表中元素的顺序依次插入到一棵初始为空的二叉排序树，画出插入完成后的二叉排序树，并求出等概率情况下查找成功的平均查找长度；
2. 若对表中的元素进行排序构成有序表，对此有序表进行折半查找，画出对其进行折半查找时的判定树，并计算出查找成功的平均查找长度；
3. 按表中元素的顺序构造一棵二叉平衡树，画出完成后的二叉平衡树，并求出等概率情况下查找成功的平均查找长度。

六、下列算法是关于堆的一些算法，读该算法并回答：

1. 什么是最小堆？什么是最大堆？（4分）
2. 在[　　] 中填上合适的语句，包括注释语句。（10分）
3. 写出最小堆的插入算法Insert(MinHeap *H，HeapData x)，x为在堆中插入的新元素。（6分）

```
//最小堆的定义
#define MaxHeapSize 100;
typedef int HeapData;
typedef struct {
    HeapData data[MaxHeapSize];   //存放最小堆元素的数组
    int Csize ; //存放当前元素个数
}MinHeap;

void InitHeap (MinHeap *H) //堆初始化，置空堆
{H->Csize = 0;}

int HeapEmpty(MinHeap *H) //  ①
    {return H->Csize ==0;}
int HeapFull(MinHeap *H)//判堆满否
{return      ②  }

void Crt_MinHeap (MinHeap *H，HeapData arr[ ]，int n) {
//根据给定数组中的数据和大小，建立堆
    for (int i =0; i<n ; i++) H->data[i]=arr[i];
    H->Csize =n ;//  ③
    int Cpos = (H->Csize-2)/2;   //最后分支结点
    while (Cpos >=0) {//从下到上逐步扩大堆
        FilterDown (&H，Cpos，H->Csize-1);
        ④ ;
    }
}

void FilterDown(MinHeap *H，int start ，int EndOf Heap){
//最小堆的向下调整算法
    int i = start，  j=2*I+1;// j是i的左子女
     HeapData temp = H->data[i];
     While (  ⑤  ){
        if (j<EndOfHeap && H->data[j]>h->data[j+1]) j++; //  ⑥
        if (temp <= H->data[j]) break;
        else {H->data[i] = H->data[j];
        i=j;   j=2*j+1;} //  ⑦
    }

void FilterUp (MinHeap *H，int start){
//最小堆的向上调整算法，从start开始，向上直到0，调整堆
    int j = start ，  i = (j-1)/2; //i是j的双亲
    HeapData temp = H->data[j];
    While (j>0){
        if (H->data[j] <= temp)   ⑧ ;
        else {H->data[j] = H->data[i];
            j = I; i =   ⑨ ;
    }
    ⑩
}
```

七、什么是内排序？什么是外排序？举例说明哪些排序方法（至少3种）是不稳定的？设待排序的排序码序列为{12，2，16，30，28，10，16*，20，6，18}，试分别写出使用2路归并、希尔、快速排序方法每趟排序后的结果。并说明每趟排序做了多少次排序码比较。（20分）

参 考 答 案

一、1．3

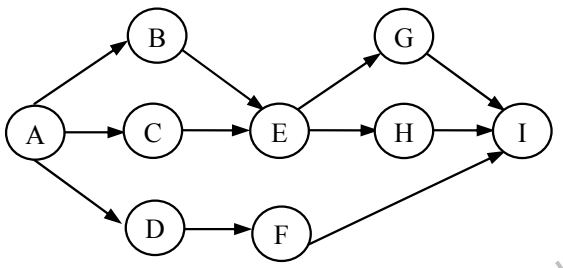
2．49，70

3．n

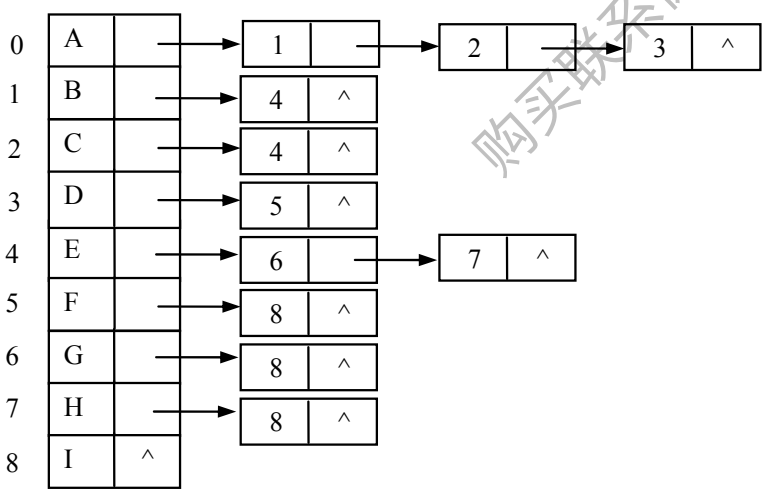
4．深度优先遍历算法

二、1.×　　2.×　　3.√　　4.×　　5.√

三、1.



2.



3. A、B、C、E、F、G、H、I（不惟一）

四、

频数	7	19	2	6	32	3	21	10
哈夫曼编码	0010	10	00000	0001	01	00001	11	0111
等长编码	000	001	010	011	100	101	110	111

对于哈夫曼编码：带权路径长度为2.6，

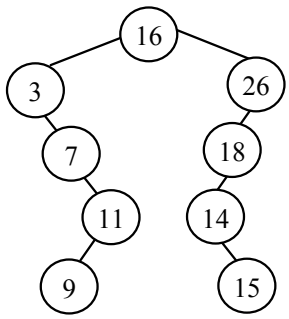
WPL={(2+3)*5+(6+7+10)*4+(32+21+19)*2}/100=2.61

对于等长编码：带权路径长度为3。

显然哈夫曼编码可以大大提高通信信道的利用率，提高报文发送速度、节省存储空间。

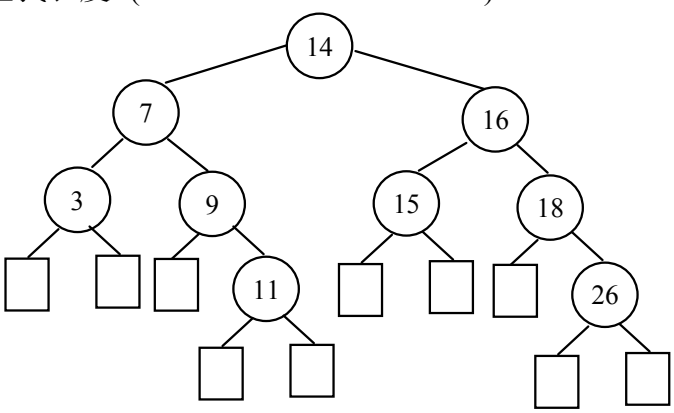
五、

1.



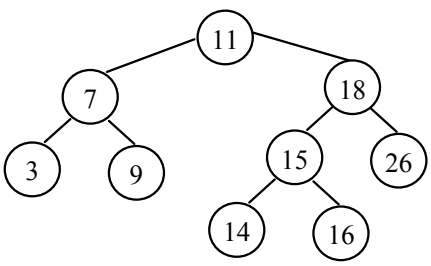
查找成功的平均查找长度=(5×2+4×2+3×2+2×2+1)/9=29/9

2.



查找成功的平均查找长度= (4×2+3×4+2×2+1)/9=25/9

3.



查找成功的平均查找长度=(4×2+3×4+2×2+1)/9=25/9

六、1. 设有一个关键字集合，按完全二叉树的顺序存储方式存放在一个二维数组中。对它们从根开始，自顶向下，同一层自左向右从0开始连续编号。若满足$K_i \leqslant K_{2i+1} \&\& K_i \leqslant K_{2i+2}$或$K_i \geqslant K_{2i+1} \&\& K_i \geqslant K_{2i+2}$，则称该关键字集合构成一个堆。前者称为最小堆，后者称为最大堆。

2. ① 判堆空否　② H->Csize==MaxHeapSize　③ 数组传送堆大小
④ Cpos--　⑤ j<=EndOfHeap　⑥ 两个子女中选小的　⑦ 向下滑动

⑧ break　　⑨ (i−1)/2　　⑩ H−>data[j]=temp;

3. 最小堆的插入算法：

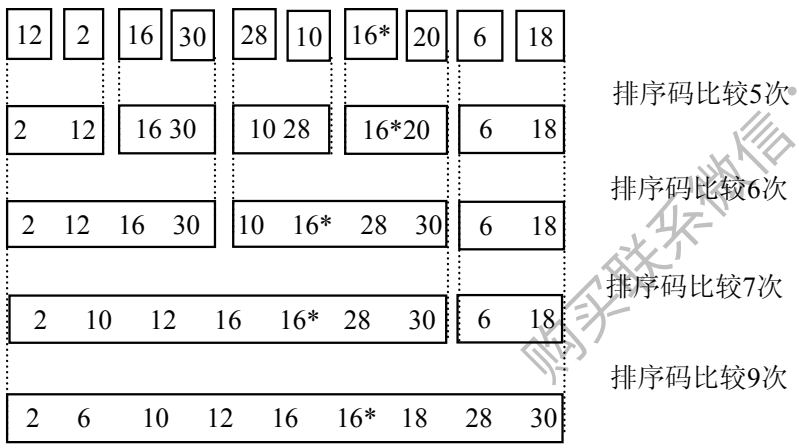
```
int Insert (MinHeap *H，HeapData x){
   if (H−>Csize==MaxHeapSize)   //堆满
   {printf("\n堆已满")}return 0;}
H−>data[H−>Csize]=x;            //插在表尾
FilterUp(&H，H−>Csize);         //向上调整
H−>Csize++;     //堆元素增一
return 1;
```

七、内排序是排序过程中参与排序的数据全部在内存中所做的排序，排序过程中无需进行内外存数据传送，决定排序方法时间性能主要是数据排序码的比较次数和数据对象的移动次数。

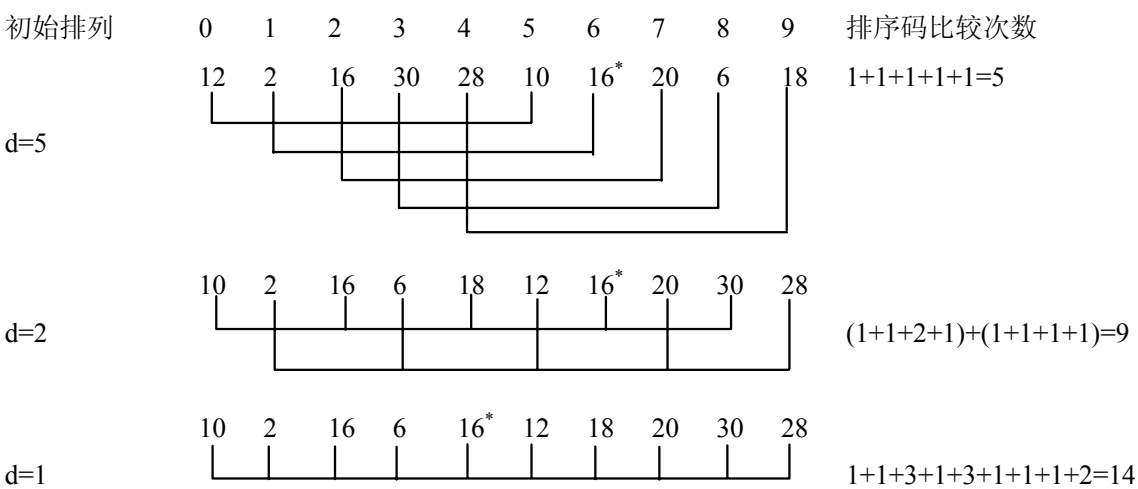
外排序是在排序的过程中参与排序的数据太多，在内存中容纳不下，因此在排序过程中需要不断进行内外存的信息传递的排序。决定外排序时间的性能主要是读写磁盘次数和在内存中总的记录对象的归并次数。

不稳定的排序方法主要有希尔排序、直接选择排序、堆排序、快速排序。

2路归并排序：



希尔排序：（增量5，2，1）



2 6 10 12 16 16* 18 20 28 30

快速排序：

Pivot	Pvtpos										
		0	1	2	3	4	5	6	7	8	9	
12	0，1，2，3	[12	2	16	30	28	10	16*	20	6	18]	9
6	0，1	[6	0	10]	12	[28	16	16*	20	30	18]	2
28	4，5，6，7，8	[2]	6	[10]	12	[28	16	16*	20	30	18]	5
18	4，5，6	2	6	10	12	[18	16	16*	20]	28	[30]	3
16	4	2	6	10	12	[16*	16]	18	[20]	28	30	1
		2	6	10	12	16*	[16]	18	[20]	28	30	

华中科技大学2002年硕士研究生入学考试试题

一、单项选择题（从下列各题四个备选答案中选出一个正确答案，将其代号（A，B，C，D）按题号次序写在答题纸上；答案不用代号，答案选错者，该题得0分；每小题1分，共12分）。

1．一个三元组表用于表示一个_____。

A．线性表　　B．广义表　　C．双向链表　　D．稀疏矩阵

2．允许对队列进行的操作有_____。

A．删除队首元素列　　B．取出最近进队的元素

C．在最早入队元素之前插入元素　　D．排序

3．设广义表上$L=(a,(((f,g),e),(c,d)))$，则表达式Tail (Head (Tail (*L*)))的值为_____。

A．(*d*)　　B．(*e*)　　C．*g*　　D．*e*

4. 假设8行10列的二维数组a[1..8，1..10]分别以行序为主序和以列序为主序顺序存储时，其首地址相同，那么以行序为主序时元素a[3，5]的地址与以列为主序时元素_____的地址相同。

A．a[5，3]　　B．a[8，3]　　C．a[1，4]　　D．答案A、B、C均不对

5．深度为5的满二叉树共有_____个分支结点。

A．32　　B．15　　C．30　　D．31

6．若树T有a个度为1的结点，b个度为2的结点，c个度为3的结点，则该树有_____个叶子结点。

A．1+2b+3c　　B．a+2b+3c　　C．2b−3c　　D．1+b+2c

7．若二叉树T的前序遍历序列和中序遍历序列分别是：b，d，c，a，e，f和c，d，e，

a，b，f，

则其后序遍历序列是_____。

A．c，e，a，d，f，b　　B．f，e，a，c，d，b
C．e，a，c，d，f，b　　D．答案A、B、C均不对

8．边数很多的稠密图，适宜用_____表示。

A．邻接矩阵　　B．邻接表　　C．逆邻接表　　D．邻接多重表

9．对22个记录的有序表作折半查找，当查找失败时，至少需要比较_____次关键字。

A．3　　B．4　　C．5　　D．6

10．查找哈希（Hash）表，不会发生冲突的哈希函数是_____。

A．除留余数法　　B．伪随机探测再散列法
C．直接地址法　　D．线性探测再散列法

11．对有序单链表使用_____查找法进行查找。

A．折半　　B．分区（分块）　　C．哈希（Hash）　　D．顺序

12．直接插入排序在最好情况下的时间复杂度为_____。

A．$O(\log_2 n)$　　B．$O(n)$　　C．$O(n\log_2 n)$　　D．$O(n^2)$

二、多项选择题（从下列各题四个备选答案中选出二至四个正确答案，将其代号（A，B，C，D）按题号次序写在答题纸上；答案不用代号、答案选错或未选全者，该题得0分；每小题2分，共8分）。

1．线性表在_____时，宜使用链接表实现。

A．需不断对其进行插入删除　　B．需经常对其进行查找
C．无足够连续存储空间　　D．其结点含大量信息

2．设依次进入一个栈的元素序列为d，a，c，b，可得到出栈的元素序列为_____。

A．d，c，b，a　　B．a，c，d，b　　C．a，b，c，d　　D．c，b，d，a

3．从n（n>100）个整数中求3个最大值，采用_____排序，所需比较关键字（整数）的次数最少。

A．简单选择　　B．归并　　C．快速　　D．冒泡

4．在最坏情况下，_____排序算法的时间复杂度为$O(n^2)$。

A．堆　　B．归并　　C．快速　　D．直接插入

三、算法填空题（按下划线上的序号次序写在答题纸上，每空1分，共18分）

1．下列C算法（程序）完成操作：1)输入一个数列，以零为结束标志，按“先进先出”

方式生成单链表；2)输出表中的结点；3)逆置表中的结点；4)排序；5)输出表中的结点，释放全部结点所占的空间。例如，设输入数列L=(10，20，5，14，0)，逆置后变为L=(0，14，5，20，10)；排序后输出：0，5，10，14，20。C算法如下：

```
main()
{struct node
      {int data;
      struct node *next;
}*head，*tail，*p，*q，*f，*min;
int x;
head=tail=    (1)    ;   /*生成带表头结点的单链表*/
do
{     p=    (2)    ;
      scanf("%d"，    (3)    );
      tail->next=p;    (4)    ;
}while (p->data);
tail->next=NULL;
p=    (5)    ;      /*输出表中的结点*/
while (p)
{     printf("%d"，p->data);    (6)    ;
      }
q=head->next; head->next=NULL;   /*逆置表中的结点*/
while (q)
{     f=q; q=q->next;
      f->next=    (7)    ;
      head->next=    (8)    ;
}
p=head->next;   /*选择排序*/
while (p->next)
{     min=p;     q=p->next;
      while (q)
      {     if (q->data<min->data)    (9)    ;
                  (10)    ;
      }
      if (    (11)    )
      {     x=p->data; p->data=min->data; min->data=x;}
                  (12)    ;
      }
      q=head->next;          /*输出结点，释放空间*/
      while (q)
      {     printf("%d"，q->data); f=q;
            q=q->next;    (13)    ;
      }
      (14)    ;
}
```

2．设n个数的数列存放在数组$a[1..n]$（下标1~n）中，下列算法将变为一个堆，注意：本算法不是完整的堆排序算法，仅将a变为堆顶元素具有最大值的“大堆”，是初始堆。

```
void adjust (in: a[ ],   int n)
{     int i，j，s，x;
      for (i=n/2; i>=1; i--)
   {        s=i; x=a[s];
            for (j=2*s; j<=n; j*=2)
            {     if (j<n && a[j]<a[j+1])_____(15)_____;
                  if (x>a[j])_____(16)_____;
                  a[s]=a[j]; s=_____(17)_____;
            }
            a[s]=_____(18)_____;
   }
}
```

四、阅读下列算法：

```
void suan_fa (int n)
{     int i，j，k，s，x;
      for (s=0，i=0; i<n;i++)
            for (j=i; j<n; j++)
                  s++;
      i=1; j=n; x=0;
      while (i<j)
      {i++; j--; x+=2;}
      printf ("s=%d",   x="%d",   s,   x);
}
```

1．分析算法中语句“s++；”的执行次数；

2．分析算法中语句“x+=2；”的执行次数；

3．分析算法的时间复杂度；

4．假定*n*=5，试指出执行该算法的输出结果。　　（共6分）

五、假定用二叉链表表示一棵二叉树：

1．试写出二叉链表的结点的类型定义；

2．设root为指向二叉树的根结点的指针，试写一算法：求二叉树中分支结点和叶子结点的数目，输出结果。（共6分）

参 考 答 案

一、

1．D	2．A	3．D	4．D
5．B	6．D	7．A	8．A
9．C	10．C	11．B	12．B

二、

1．ACD　　2．ABC　　3．BC　　4．CD

三、

1. （1）(struct node *) malloc (sizeof (struct node))
（2）(struct node *) malloc (sizeof (struct node))
（3）p->data
（4）tail=p
（5）head->next
（6）p=p->next
（7）head->next
（8）f
（9）min=q
（10）q=q->next
（11）p->data<min->data
（12）p=p->next
（13）free (f)
（14）free (head)

2. （15）j=j+1
（16）break
（17）j
（18）x

四、

1. “s++;”语句的执行次数为：$1+2+\cdots+n=\frac{n(n+1)}{2}$。

2．“x+=2;”语句的执行次数为$\left\lfloor\frac{n}{2}\right\rfloor$。

3. 在for循环语句中时间复杂度为$\frac{n(n+1)}{2}$，在while循环语句中时间复杂度为$\left\lfloor\frac{n}{2}\right\rfloor$，所以，算法时间复杂度为$O(n^2)$。

4. s=15，x=4

五、

1. 树结点类型定义为：

```
typedef struct node {
        ElemType data;
        struct node *lchild;
        struct node *rchild;
}BTree
```

```
2. static int a[2]          //用于存储分支结点与叶子结点数目
   void count node (BTree *b)
   {
      if (b==Null) return;
      if (b->lchild==Null && b->rchild==Null) a[0]=a[0]+1;
      else a[1]=a[1]+1;
      count node (b->lchild);
      count node (b->rchild);
   }
   main ( )
   {
      BTree *root;
      count node (root);
      cout <<a[1]<<"    "<<a[0]<<endl;
   }
```

南开大学2002年硕士研究生入学考试试题

一、选择题（每小题3分，共15分）

在下列各题中，每题之后均有若干个备选答案，请选出所有正确的答案，填入“_____”处。答案请写在答题纸上。

1．有n个记录的文件，若关键字位数为d，基数为r，则基数排序共需进行____遍分配与收集。

A. n　　B. r　　C. d　　D. $d+r$

2．在有向图的邻接表存储结构中，顶点v在表结点中出现的次数等于_____。

A. 顶点v的度　　B. 顶点v的出度
C. 顶点v的入度　　D. 依附于顶点v的边数

3．散列函数有一个共同性质，即函数值应按_____取其值域的每一个值。

A. 最大概率　　B. 最小概率
C. 同等概率　　D. 平均概率

4．设线性表的每个元素占8个存储单元。第1个单元的存储地址为100，则第6个元素占用的最后一个存储单元的地址为_____。

A. 139　　B. 140　　C. 147　　D. 148

5．直接插入排序在最好情况下的时间复杂度为_____。

A. $O(\log n)$　　B. $O(n)$　　C. $O(n\log n)$　　D. $O(n^2)$

二、填空题（共8分）

1. （1分）在带表头结点的单链表中，当删除某一指定结点时，必须找到该结点的结点。
2. （3分）列出图的三种存储表示方式_____、_____和_____。
3. （2分）图的遍历算法有_____和_____。
4. （2分）用S表示入栈操作，X表示出栈操作，若元素入栈顺序为1234，为了得到1342出栈顺序，相应的S、X操作串为_____。

三、（7分）设有三对角矩阵$(A_{ij})_{n\times n}$，将其三对角线上元素按行存于一维数组B[1..m]中，使B[k]=A[i，j]，

求：（1）请写出用i，j表示k的下标计算公式

（2）请写出用k表示i，j的下标计算公式

如五阶三对角矩阵的形式如下：

$$\begin{bmatrix} \times & \times & & & \\ \times & \times & \times & & \\ & \times & \times & \times & \\ & & \times & \times & \times \\ & & & \times & \times \end{bmatrix}$$

，其中，三对角线之外的数全为0。

四、（8分）对下列数据表，写出采用希尔排序算法排序的每一趟结果。

（100，12，20，31，1，5，44，66，61，200，30，80，150，4，8）

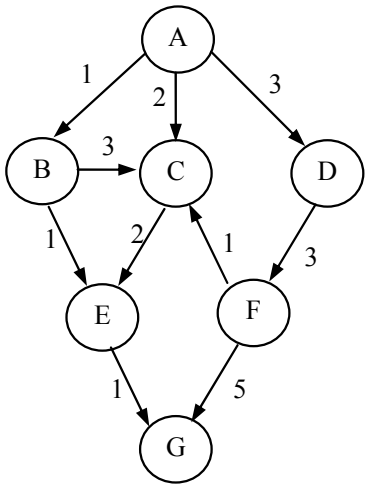
设增量序列为：d={5，3，1}

五、（10分）什么是哈夫曼树？简述哈夫曼编码过程。试证明有n个叶子的哈夫曼树共有$2n-1$个结点。

六、（10分）设有一棵二叉树T，任意一条从根到叶子结点的路径将T分成三部分，在路径左边的结点组成集合S_1，路径上的结点组成集合S_2，路径右边的结点组成集合S_3。现要求以三个带表头的单链表L_1、L_2及L_3分别存储这三个集合中的元素，其中L_i存储集合S_i中的元素（i=1，2，3）。试设计二叉树T及单链表L_1、L_2及L_3的数据结构，并用类C程序设计语言设计算法实现上述功能（不必使用C或C++语言）。

七、（6分）顺序给出以下关键字：65、23、31、26、7、91、53、15、72、52、49、68，试画出建立的二叉排序树。再从建好的二叉排序树中依次删除关键字91、65，试分别画出删除关键字后的二叉排序树。

八、（6分）对如下带权有向图，



求（1）每个顶点的入度和出度；

（2）邻接矩阵；

（3）邻接表。

参 考 答 案

一、

1．C　　2．C　　3．C　　4．C　　5．B

二、

1．前驱

2．邻接矩阵　邻接表　十字链表

3．深度优先搜索遍历算法　广度优先搜索遍历算法

4．SXSSXSXX

三、

（1）$k=2i+j+1$

（2）$i=\left\lfloor \frac{k-1}{3} \right\rfloor$，$j=k-1-2=\left\lfloor \frac{k-1}{3} \right\rfloor$

四、

d=5：5，12，20，4，1，30，44，66，31，8，100，80，150，61，200

d=3：4，1，20，5，12，30，8，61，31，44，66，80，150，100，200

d=1：1，4，5，8，12，20，30，31，44，61，66，80，100，150，200

五、

哈夫曼树又称最优二叉树，它是n个带权叶子结点构成的所有二叉树中，带权路径长度

*WPL*最小的二叉树。

哈夫曼编码过程：

（1）根据给定的n个权值$\{w_1, w_2, \cdots, w_n\}$构成$n$棵二叉树的集合$F=\{T_1, T_2, \cdots, T_n\}$，其中每棵二叉树$T_i$（$1 \leqslant i \leqslant n$）都只有一个权值为$w_i$的根结点，其左右子树均为空。

（2）在F中选出两棵根结点的权值最小的树作为左右子树构造一棵新的二叉树，且置新树根结点的权值为其左、右子树根结点的权值之和。

（3）从F中删除这两棵树，同时将新得到的二叉树加入F中。

（4）重复第（2）步和第（3）步，直到F中只含有一棵树为止，此树便是哈夫曼树。

证明有n个叶子结点的哈夫曼树共有$2n-1$个结点。

证明：因为哈夫曼树为二叉树，设其分支结点数为n，由二叉树的性质：二叉树终端结点数等于双分支结点数加1，

哈夫曼树中的分支结点都为双分支结点（此处可由哈夫曼算法得出），

所以$n=n_1+1$

$n_1=n-1$

哈夫曼树共有$n+n_1=n+n-1=2n-1$个结点。

六、

二叉树T用二叉链表存储：

```
typedef struct BiTNode {
      TelemType data;
      struct BiTNode *lchild,*rchild;
}
BiTNode *BiTree;
```

单链表L_i的存储结构：

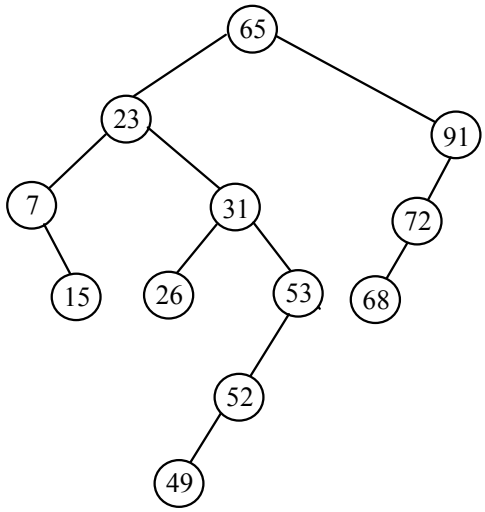
```
typedef struct LNode {
       ElemType data;
       struct LNode *next;
}
LNode *linklist;
```

类C语言实现题中要求的功能的算法如下：

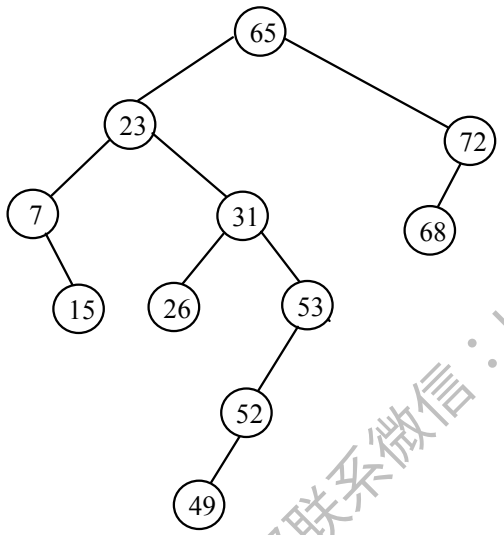
```
main divide (BiTree root, linklist L1,L2,L3)
{ duiding(root);
}
duiding (BiTree p)
{ 将当前树的根结点存入L2;
  if   根结点的左孩子在路径上
  then  把根结点的右子树全部存入L3中
          cluiding (root↑.lclikld)
  else  把根结点的左子树全部存入L1
        duicling(root↑.rchild)
}
```

七、

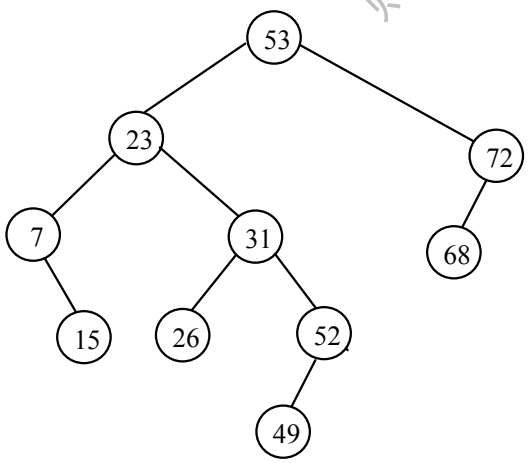
建立的二叉排序树为：



删除91后的二叉排序树为：



删除65后的二叉排序树为：



八、（6分）

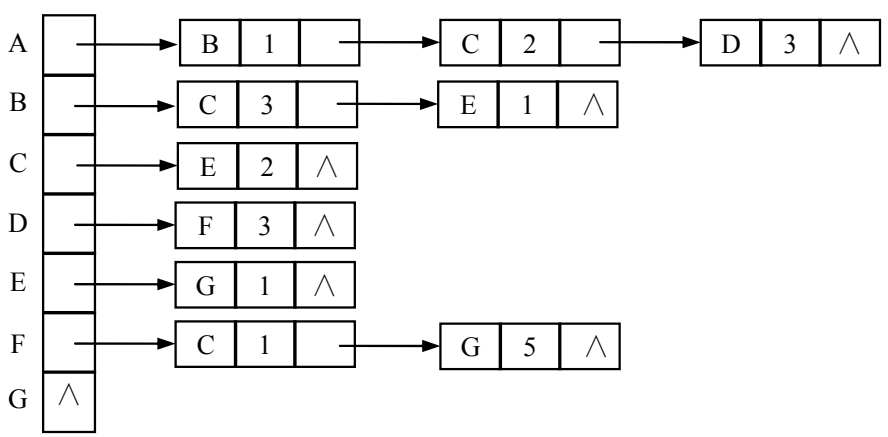
（1）A结点的入度为0，出度为3
B结点的入度为1，出度为2
C结点的入度为3，出度为1

D结点的入度为1，出度为1
E结点的入度为2，出度为1
F结点的入度为1，出度为2
G结点的入度为2，出度为0

（2）邻接矩阵如下：

	A	B	C	D	E	F	G
A	0	1	2	3	∞	∞	∞
B	∞	0	3	∞	1	∞	∞
C	∞	∞	0	∞	2	∞	∞
D	∞	∞	∞	0	∞	3	∞
E	∞	∞	∞	∞	0	∞	1
F	∞	∞	1	∞	∞	0	5
G	∞	∞	∞	∞	∞	∞	0

（3）邻接表如下：



南京大学2003年硕士研究生入学考试试题

一、数据结构算法分析题（10分，每空2分）

1．设图G，其顶点数为n，边数为e；
对用邻接矩阵表示的图G进行任何一种遍历时，其时间复杂度为_____。
对用邻接表示的图G进行任何一种遍历时，其时间复杂度为_____。

2．斐波那契（Fibonacci）数列F_n定义如下：
$F_0=0$，$F_1=1$，$F_n=F_{n-1}+F_{n-2}$，$n=2$，3，…

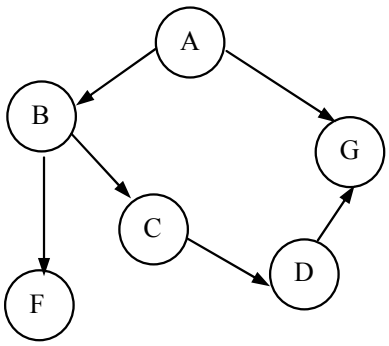
给出递归计算F_n时，递归函数的时间复杂度为_____。

3．设栈S和队列Q的初始状态为空，元素a_1，a_2，a_3，a_4，a_5，a_6，a_7和a_g依次通过栈S，一个元素出栈后立即进入队列Q，若8个元素出队列的顺序是a_3，a_6，a_7，a_5，a_8，a_4，a_2，a_1，则栈S的容量至少应该是多少（即至少应该容纳多少个元素）_____。

4．一棵含有n个关键字的m阶B−树中进行查找，至多读盘_____次。

二、解答题（每题5分，共10分）

1．画出以下有向图的存储数组；并画出使用递归算法对以下有向图进行深度优先搜索时所用栈的变化情况（若当前结点有多个未访问邻居，则按逆时针顺序来访问这些邻居）。



2．采用合适的数据结构设计一个算法来求出n个元素a_1，a_2，…，a_n的所有组合，如{}、{a_1，a_2}、{a_2，a_3，a_4}等，要求给出所用算法的详细描述。

三、数据结构算法设计题（共20分）

1．（10分）设二叉树T是右线索化的中序线索树（即若一个结点的右子树为空，则对应的rightchild指向它的中序下的后继结点），P指向T中除根结点以外的某个结点，试给出找P所指结点的父母结点的算法。

要求写一个完整的右线索树的复合类说明，并将该算法作为其成员函数且写出其具体的实现（请用C++语言来编写）。

2．（10分）对于待排序序列{12，11，13，49，26，14，8，7}：

（1）以快速排序算法来将该序列进行排序，写出各趟排序后的结果；

（2）以该序列为输入序列来建立平衡二叉搜索树（即AVL树），并求出其搜索成功的平均搜索长度ASL_{succ}；

（3）设该平衡二叉树的物理结构采用链表存储表示，每个结点包括三个域：数据data、左子女结点指针leftChild和右子女结点指针rightChild，写出使用栈来中序遍历该平衡二叉树的非递归算法（以C语言或者C++语言描述）。

参 考 答 案

一、

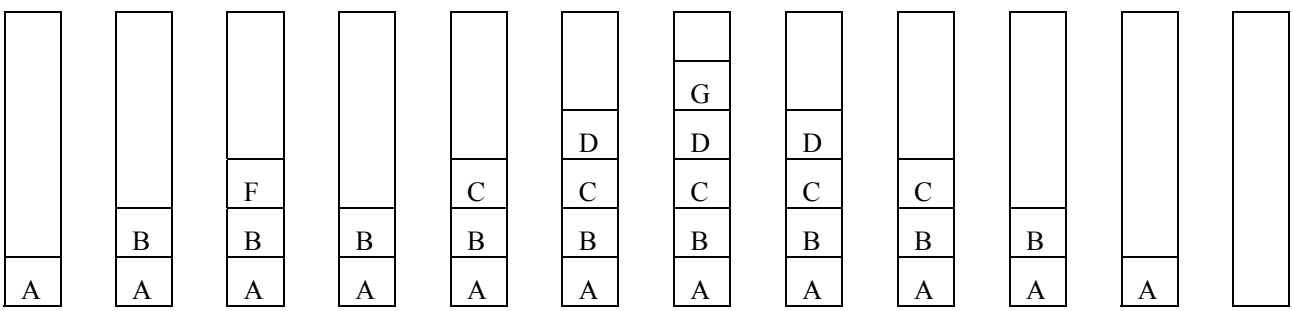
1．$O(n^2)$　$O(e)$　　2．$O(2n)$　　3．5　　4．$1+\log_{\lceil m/2 \rceil}(n+1)/2$

二、

1．存储数组为（2分）

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

栈的变化情况如下（3分）：



（1分）　　（1分）　　（1分）

2．算法主要有两种

1）递归算法：

采用线性结构（特别是数组）作为数据结构（1分）

```
void Combination(int i,  int n){
//进入函数时已经对前i-1个元素作了取舍处理，现从第i个元素起进行取舍。
    if(i>n)输出一个组合；
    else{取第i个元素；Combination(i+1，n);
        舍第i个元素；Combination(i+1，n);
    }// if
}//Combination
```

2）非递归算法：

使用位向量来表示组合，如$\{a_1, a_2, a_4\}$用1101000…表示，$\{a_2, a_3, a_n\}$用0110…1表示（1分）。

算法描述有如下两种（4分）

a．位向量的初值为000…00；

```
while（向量中存在值为0的位时）{
  从右到左找出第一位0，设其位置为i；
  将第i位变为1，且将第i位到第n位全部变为0；
  根据当前的位向量来输出一种组合；
}
```

或者：

b．位向量的初值为000…00；

```
while（向量不为111…11时）{
   将当前向量以加1（二进制加法）；
   根据当前的位向量来输出一种组合;
}
```

三、

1．算法如下

```
class ThreadTree;
class ThreadNode;
{
     frend class ThreadTree;
     public:
          ThreadNode(const int item):data(item)，leftchild(NULL)，rightchild(NULL)，
                rightThread(0){}
      private:
          int data;
          ThreadNode *firstchild，*rightchild;
          Int rightThread;

      };

  class ThreadTree
    {
         public:
            ...
            ThreadNode*parent(ThreadNode*p);
            ThreadNode*Findparent(ThreadNode*t，ThreadNode*p);          （3分）
     };
  ThreadNode*ThreadTree::parent(ThreadNode*p)
    {
     if(root==NULL){return NULL;}
     ThreadNode*t=root;
      Return Findparent(t，p);
    }
  ThreadNode*ThreadTree::Findparent(ThreadNode*t，ThreadNode*p)
   {
      if(t=p)return NULL;
      ThreadNode*q=p;*s;
      while(q->rightThread==0)q=q->rightchild;                          （3分）
      if(q->rightchild==NULL)
         { s=t;
            while(s->rightchild!=p)s=s->rightchild;                     （2分）
          }
```

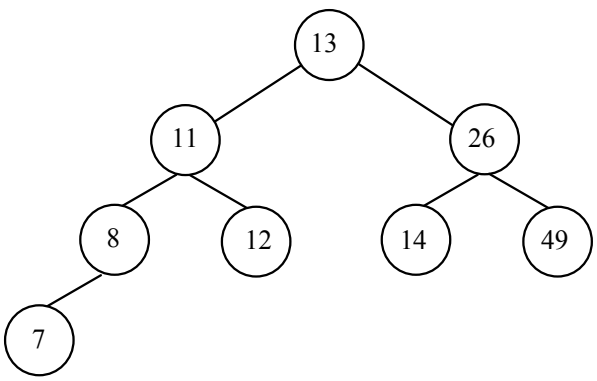
```
        else
           {s=q->rightchild;
             if(s->leftchild!=p)
                 { s=s->leftchild;
                     while(s->rightchild!=p)s=s->rightchild;                    （2分）
                 }
            return s;
        }
```

2.

（1）快速排序算法中各趟排序的结果为（3分）：

第一趟排序结果：7，11，8，12，26，14，49，13
第二趟排序结果：7，11，8，12，13，14，26，49
第三趟排序结果：7，8，11，12，13，14，26，49；

（2）平衡二叉搜索树为：（3分）



其搜索成功的平均搜索长度为(1+2×2+3×4+1×4)/9=7/3　　（1分）

（3）参考代码，其中S代表栈，T代表树（3分）：

```
InitStacks(S);Push(S，T);
while(!StackEmpty(S)){
    while(GetTop(S，p)&& p)Push(S，p->leftChild);
    If(!StackEmpty(S)){
       Pop(S，p);
       if(!Visit(p->data))return ERROR;
       Push(S，p->rightChild);
    }//if
}//while
```

或者

```
InitStack(S);p=T;
while(p| |StackEmpty){
    if(p){Push(S，p);p=p->leftChild}
    else{
       Pop(S，p);if(!Visit(p->data))return ERROR;
       p=p->rightChild;
     }//else
}//while
return OK;
```

武汉理工大学2003年硕士研究生入学考试试题

一、选择题（答案可能不惟一，共26分）

1．下列各项中属于逻辑结构的有_____。

A．哈希表　　B．有序表
C．单链表　　D．顺序表

2．由3个结点组成的二叉树的深度可能是_____。

A．1　　B．2
C．3　　D．4

3．将一个A[1..100，1..100]的三对角矩阵以行序为主序存入一维数组B[1..298]中，元素a[66，65]在B数组中的位置*k*等于_____。

A．198　　B．197
C．196　　D．195

4．一棵满二叉树同时又是一棵_____。

A．完全二叉树　　B．二叉排序树
C．正则二叉树　　D．平衡二叉树

5．长度为*n*的顺序存储的线性表，在任何位置上插入或删除一个元素的概率相等，插入一个元素时平均需移动_____个元素，删除一个元素时平均需移动_____个元素。

A．(*n*+1)/2　　B．*n*/2
C．(*n*−1)/2　　D．(*n*−2)/2

6．用s表示入栈操作，*表示出栈操作，栈的初态、终态均为空，入栈和出栈的操作序列可表示为仅由s和*组成的序列，下面给出的序列中合法的操作序列有_____。

A．s*ss*s**　　B．sss**s**
C．s**s*ss*　　D．sss*s*s*

7．_____是特殊的线性表。

A．队列　　B．哈希表
C．栈　　D．判定表

8．表长为25的哈希表，用除留余数法，即按公式H(key)=key mod p建立哈希函数，则p应取_____为宜。

A．23　　B．24

C．25　　D．26

9．任一个有向图的拓扑序列_____。

A．可能不存在　　B．有一个
C．一定有多个　　D．有一个或多个

10．在下列排序方法中，_____方法可能出现这种情况：在最后一趟开始之前，所有的元素都不在其最终应在的正确位置上。

A．快速排序　　B．冒泡排序
C．堆排序　　D．插入排序

11．若以[4，5，6，7，8]作为权值构造Huffman树，则该树的带权路径长度为_____。

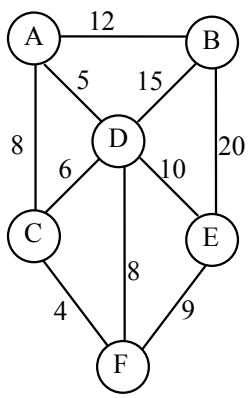
A．67　　B．68
C．69　　D．70

12．设head(L)、tail(L)分别为取广义表表头、表尾的操作，则从广义表L=((x，y，z)，a，(u，v，w))中取出原子u的运算为_____。

A．head(tail(tail(head(L))))　　B．tail(head(head(tail(L))))
C．head(tail(head(tail(L))))　　D．head(head(tail(tail(L))))

二、填空题（共32分）

13．在单链表中设置头结点的作用是_____。
14．线索二叉树的左线索指向其_____，右线索指向其_____。
15．树在计算机内的链式存储表示方法有_____、_____和_____。
16．_____现象称为“冲突”。
17．用（A，B）的形式表示边，对下图的无向图，从顶点A开始求最小生成树，用Prim算法产生的边依次是_____，用Kruskal算法产生的边依次是_____。



18．用数组a[1..n]表示循环队列S_q，当循环队列S_q满时，队列中共有_____个元素。
19．判定树常用来表示_____的查找过程。
20．直接插入排序、堆排序算法的时间复杂度分别是_____、_____。

21．树根的层次是1，则深度为8的完全二叉树至少有_____个结点，拥有100个结点的完全二叉树的最大层次数是_____。

22．对n个记录进行2路归并排序，一共需要进行_____趟归并。

三、简答题

23．有5个元素，其入栈次序为A，B，C，D，E，在各种可能的出栈序列中，第一个出栈元素为C且第二个出栈元素为D的出栈序列有哪几个？

24．一棵树的边的集合为{(I，M)，(I，N)，(E，I)，(B，E)，(B，D)，(C，B)，(G，J)，(G，K)，(A，G)，(A，F)，(H，L)，(A，H)，(C，A)}。

（1）试画出这棵树；
（2）哪个是根结点？哪些是叶子结点？
（3）树的深度是多少？
（4）将此树转化为相应的二叉树。
（5）将转化所得的二叉树进行后序穿线，建立后序线索树。

25．用十字链表存储稀疏矩阵的非零元，试画出存放非零元的结点的结构示意图，并说明结点中各个域中分别存放什么内容？

26．对n个顶点的有向图，采用邻接矩阵和邻接表表示时，如何判别下列问题：

（1）图中有多少条边？
（2）任意两个顶点i和j是否有边相连？
（3）任意一个顶点的度是多少？

27．依次输入序列（62，68，30，61，25，14，53，47，90，84）中的元素，生成一棵二叉排序树。

（1）画出生成后的二叉排序树；
（2）假定每个元素被查找的概率相等，试计算该二叉排序树的平均查找长度。

四、算法设计题（共45分）

说明：① 用类C或类Pascal语言编写算法。
② 应对算法中使用的数据类型给出必要的说明和注释。

28．用带表头结点的循环单链表作为队列的存储表示，不设头指针而只设一个尾指针rear指向队尾结点。试写出该链式队列的出队算法。

29．二叉树采用二叉链表作为存储结构，写一递归算法计算二叉树的深度。

30．已知（r_1，r_2，…，r_n）是一个小堆顶，试写一算法，使得增加一个元素r_{n-1}后，(r_1，r_2，…，r_n，r_{n-1})仍是一个堆。

参 考 答 案

一、

1．B	2．BC	3．D	4．ACD
5．B，C	6．AB	7．AC	8．A
9．A	10．D	11．C	12．D

二、

13．可以对第一元素结点无需作特殊处理，且使空表和非空表的处理相统一

14．前驱　　后继

15．双亲表示法　孩子表示法　孩子兄弟表示法

16．对不同的关键字得到同一哈希地址

17．(A，D) (D，C) (C，F) (F，E) (A，B)　(C，F) (A，D) (C，D) (F，E) (A，B)

18．$n-1$

19．折半查找

20．$O(n^2)$　　$O(n\log n)$

21．128　　7

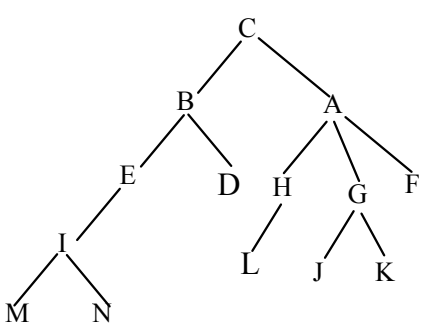
22．$\lceil \log_2 n \rceil$

三、

23．有3个：CDBAE，CDEBA，CDBEA。

24．

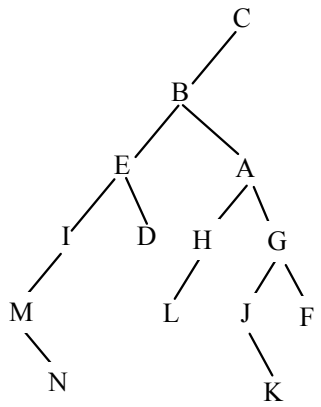
（1）



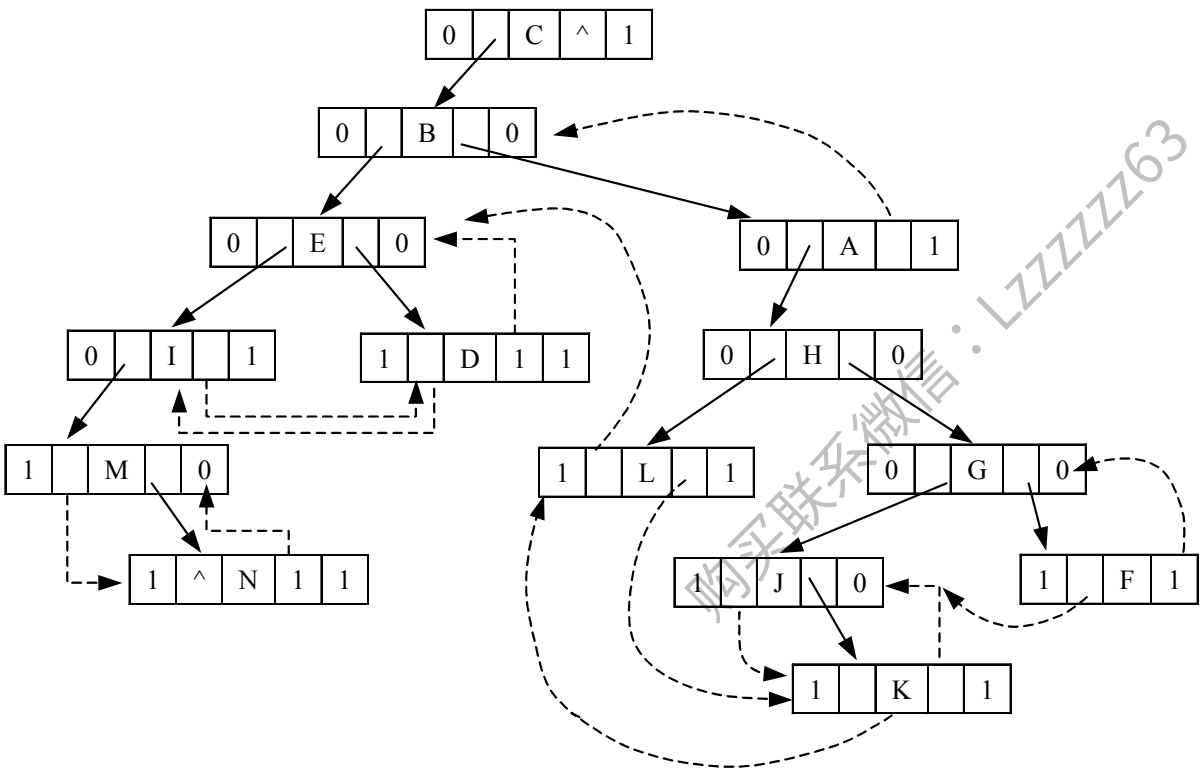
（2）根结点为C时叶子结点为M、N、D、L、J、K、F

（3）树的深度是5

（4）转化为相应的二叉树如下所示：

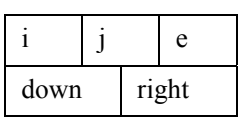


（5）后序线索二叉树的表示法：



25.

结点的结构为：



类型为

```
struct OLNodef{
    int i，j;//非零元的行和列下标
    ElemType e;//非零元的值
    Struct OLNode *down，*right;//该非零元所在行表和列表的后继链域
}
```

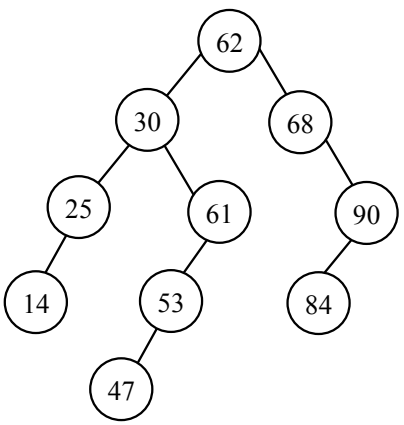
26．对于采用邻接矩阵表示的有向图：

（1）邻接矩阵中非零元素的个数即为图的边数；
（2）看第*i*行，第*j*列的值与第*j*行第*i*列的值，若存在一个不为0，则有边相连；
（3）顶点*i*的度等于第*i*行和第*j*列中非零元素个数的和。

对于采用邻接表表示的有向图：

（1）遍历整个邻接表，求出表结点的个数即为图的边数；
（2）遍历第*i*个链表和第*j*个链表中的表结点，分别查看是否存在邻接点域值为*i*和邻接点域值为*j*的表结点，若有，则有边相连；
（3）遍历整个邻接表，找出邻接点栈的值为*i*的结点数得出其入度，遍历第*i*个链表中的表结点求出其个数为顶点*i*的出度，顶点*i*的度为入度和出度之和。

27．（1）二叉排序树如下所示：



（2） $ASL=\frac{1}{10}(1+2\times2+3\times3+4\times3+5)=\frac{31}{10}$

四、

28．将循环单链表的表头作为队列的头指针即可实现。

```
Struct LNode{                              //定义结点的结构
        ElemType data;
        Struct LNode *next;
}
struct LinkQueue{
        struct LNode *rear;表尾指针
}
sturct LNode *Head;链表的头指针
ElemType QDelete(Link Queue & HQ)
{
        if(Head==NULL)
        { printf("Linked queue is empty\n");//若链队为空，则终止
                exit(1)
            }
```

```
        ElemType temp=Head->data;          //暂存队首元素以便返回
        LNode *p=Head;
        Head=Head->next;                   //头指针指向下一个元素
        if(Head==NULL)HQ.rear=NULL;
        HQ.rear->next=Head;                //修改队尾指针中的next使其成循环队列
        free(p);
        return temp;
}
```

29.

```
int BTree Depth(BTreeNode *BT)        //求由BT指针指向一棵二叉树的深度
{   if(BT==NULL)return 0;             //对于空树返回0
    else{
        int dep1=BTreeDepth(BT->left);  //计算左子树的深度
        int dep2=BTreeDepth(BT->right); //计算右子树的深度
        if(dep1>dep2)return dep1+1;
        else return dep2+1;
    }
}
```

结点的结构为：

```
struct BTreeNode {
        ElemType data;
        struct BTreeNode *left;
        struct BTreeNode *right;
}
```

30.

```
ElemType R[Maxsize] 表示-连续空间
oppend (ElemType R[ ]，int n)
{
        ElemType x;
        int i，j;
        x=R[n+1];
        i=(n+1)/2;
        j=n+1;
        while((x<R[i])&&(i>=1))
        {
            R[j]=R[i];
            j=i;
            i=i/2;
        }
        R[j]=x;
}
```

复旦大学2003年硕士研究生入学考试试题

一、填空题

1．设环形队列存放在数组q中，数组q的长度为n，下标从0~n−1。队头指针head指向队头结点，队尾指针tail指向队尾结点后一个空闲结点。
该环形队列的队空标志为_____，队满标志为_____，该队列的长度为_____。

2．设数组a[n][n]是一个二维数组，数组中存放的是下三角矩阵。所谓下三角矩阵是指主对角线的右上方元素都是零的方阵。现将a中下三角的元素（包括主对角线）按行优先存储于一维数组b中（数组a、b中下标从0开始计）。设数组a中某下三角元素为a[i][j]，存放在数组b[k]中，则k=_____。如果a中下三角元素按列优先存储在一维数组中，则k=_____。

3．设有算术表达式x+a*(y−b)−c/d，该表达式的前缀表示为_____。后缀表示为_____。

4．对关键码序列FBJGEAIDCH进行升序排列，则堆排序时，初始建堆结果的序列为_____。设关键码序列中有n个元素，则堆排序的平均执行时间和需附加的存储结点分别为_____。

5．在树转换成二叉树形式时，二叉树中每个结点的左子结点是它原来（树中）的_____结点，二叉树中每个结点的右子结点是它原来（树中）的_____结点。

6．有n个顶点的有向强连通图最多有_____条边，最少有_____条边。
有n个顶点的无向连通图最多有_____条边，最少有_____条边。

7．在m阶B−树中，除了根和叶子外，每个结点的子结点数目的范围在_____之间，同时，具有k个子结点的非叶子结点含有_____个键值。
而在m阶B+树中，具有k个子结点的结点含有_____个键值。

8．G是一个非连通无向图，共有28条边，则该图至少有_____个顶点。

二、下面是一个链接存储线性表（带“表头结点”）的选择排序函数。排序过程中，依次选出键值从小到大的结点可顺序组织成有序链表，即每次选出的键值最小的结点接在已部分完成的有序链表的末尾。请在下划线序号处填上适当内容，每处只填一个语句。

```
typedef struct node { char data;
                      struct node *link;
                    }NODE;
NODE *select_sort (NODE *h)
{    NODE *tail，*u，*v，*p;
     tail=h;
     while (tail->link!=NULL)
          {for (u=tail->link，p=tail;u->link!=NULL;  (1)  )
               if(u->link->data<p->link->data)  (2)  ;
               if(p!=tail)
                    {v=p->link; p->link=v->link;
                                  (3)  ;  (4)  ;
                    }
```

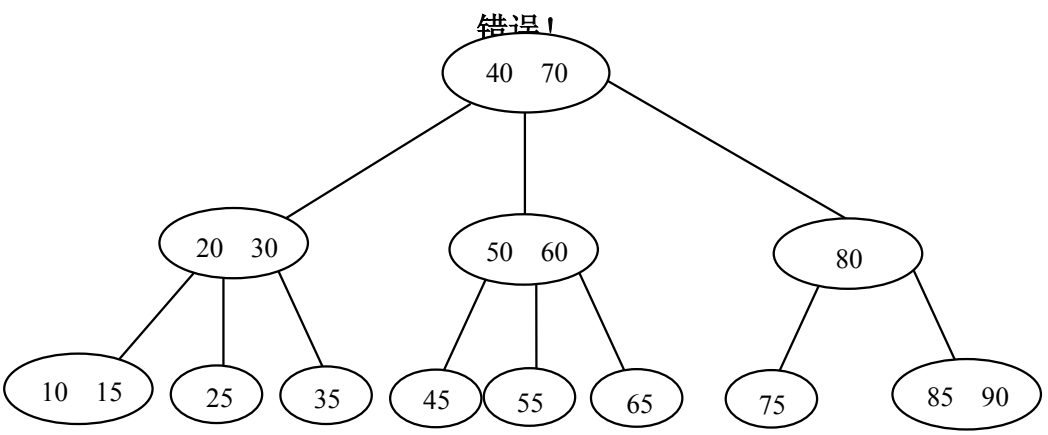
```
                    (5)    ;
            }
        return h;
}
```

三、已知一棵二叉树的前序遍历的结点序列为ABCDEFG，中序遍历的结点序列为BDCAGFE，试画出这棵二叉树，并写出后序遍历时得到的结点序列。

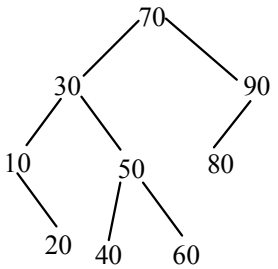
四、下面的递归函数是根据二叉树的前序遍历结果和中序遍历结果产生一棵二叉树。前序遍历结果在形参pstr中，中序遍历结果在形参istr中。产生的二叉树作为函数值返回。试在下划线的序号处填上适当内容。

```
typedef struct node{ char data;
                     struct node *lchild;
                     struct node *rchild;
                   }NODE;
NODE *restore (char *pstr，char *istr， int n)
{   NODE *ptr;
    char *rstr;
    int k;
    if(n<=0)return NULL;                    /*n为字符串长度*/
    ptr=(NODE *)malloc (sizeof (NODE));
    ptr-> data=*pstr;
    for (rstr=istr;rstr<istr+n; rstr++)
        if(*rstr=*pstr)  ____(1)____；/*在中序遍历中找根结点*/
    k=____(2)____;
    ptr->lchild=restore(____(3)____);
    ptr->rchild=restore(____(4)____);
    return ptr;
}
```

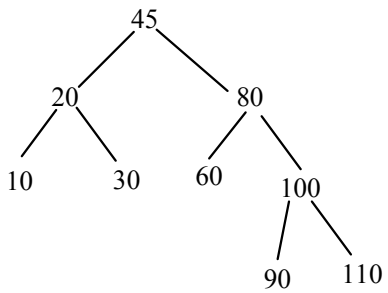
五、下图为3阶B-树的简图，若在此树上依次插入关键字88、12，试分别画出B树的变换过程。



六、下面给出两棵平衡查找树及相应的插入键值。试分别用Adelson插入算法执行相应插入后的平衡查找树。



插入55 → 答案画在答题纸上。



插入95 → 答案画在答题纸上。

七、下面是用广度优先搜索法遍历图的函数。请在下划线空白处填上适当内容，每处只填一个语句。

```
#define   MAXN 50
typedef struct L_NODE   {int ver;
                          struct L_NODE *link;
                         }L_NODE;
L_NODE *head[MAXN];              /*邻接表*/
Int visit[MAXN];

void bfs(u)
int u;
{struct queuetype{int qa，qe;          /*队列首指针、尾指针*/
                  int item[MAXN];    /*队列数组*/
                 }
typedef struct queuetype QTYPE;
int v，w;
L_NODE *T;
QTYPE queue;
printf("%4d"，u);visit[u]=1;
queue.qa=0; queue.qe=0; queue.item[0]=u;
while(    (1)    )
    {v=     (2)     ;
     t=head[v];
     while (t!=NULL)
        {w=t->ver;
         if(visit[w]==0)
              {printf("%4d"，w);
               visit[w]=1;
                  (3)     ;
```

```
            }
            (4)          ;
        }
    }
}
```

参 考 答 案

一、填空题

1. head=tail　　(tail+1)%*n*=head　　*n*−1

2. $\frac{i(i+1)}{2}+j$　　$\frac{j(j+1)}{2}+i$

3. −+x*a−yb/cd　　xayb−*+cd/−

4. ABCDEFGHIJ　　4*n*

5. 孩子　　兄弟

6. *n*(*n*−1)　　*n*　　*n*(*n*−1)/2　　*n*−1

7. $\left\lceil \frac{m}{2} \right\rceil \sim m$　　*k*−1　　*k*

8. 9

二、

（1）u=u−>link

（2）p=u

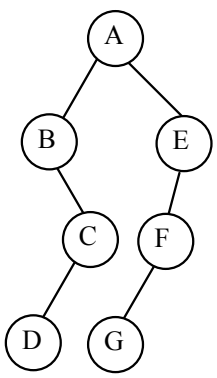
（3）v−>link=tail−>link

（4）tail−>link=v

（5）tail=tail−>next

三、后序遍历的结点序列为：DCBGFEA

二叉树如下：



四、（1）break

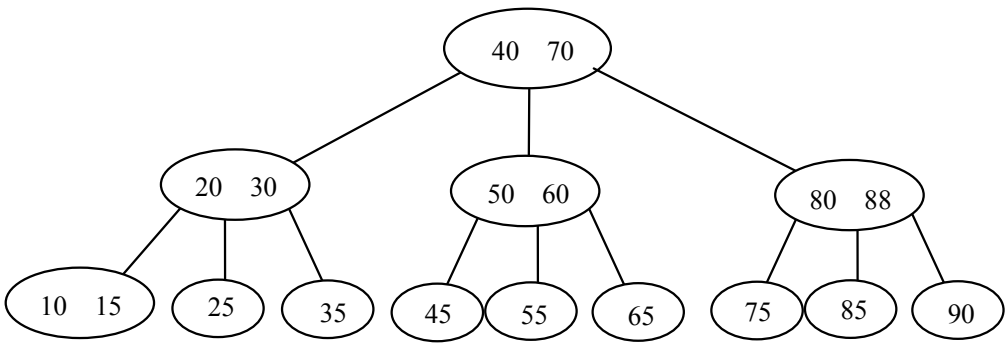
（2）rstr−istr

（3）pstr+1，istr，k

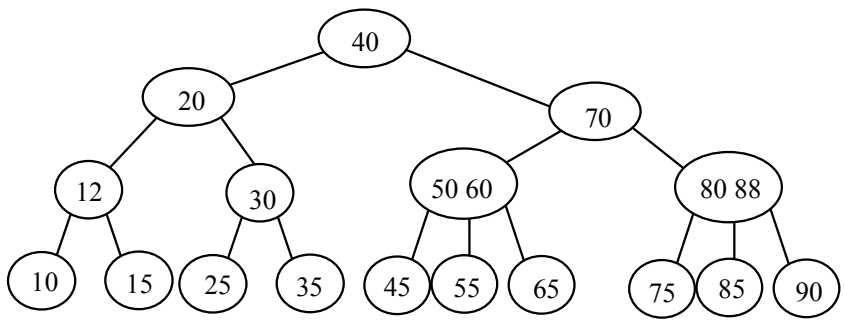
（4）pstr+k+1，istr+k+1，n−k−1

五、

插入88后

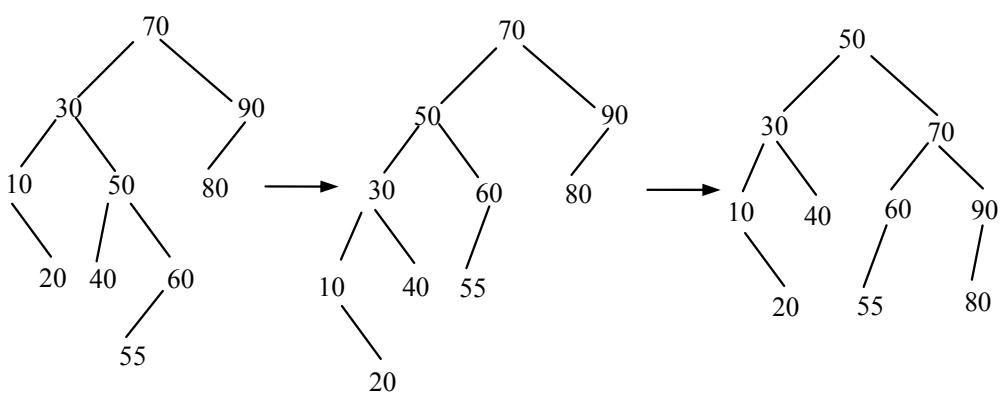


插入12后

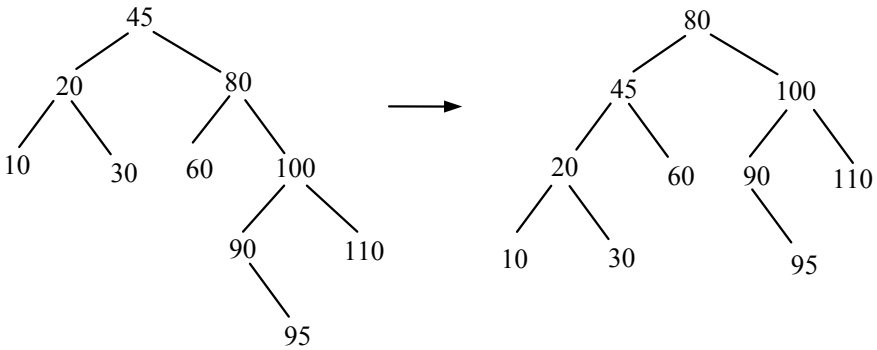


六、

插入55



插入95



七、

（1）queue.qe<=queue.qa
（2）queue.item[queue.qe++]
（3）queue.item[++queue.qa]=w
（4）t=t->link

华东师范大学2003年硕士研究生入学考试试题

一、多重选择填空题（每小题4分，共28分。每道小题都可能有一个以上的正确选项，须选出所有的正确选项，不答不得分，多选、少选或选错都将按比例扣分。请将答案写在答题纸上）。

1．链表不具有的特点是（　　）。

A．可随机访问任一个元素　　B．插入和删除时不需要移动元素
C．不必事先估计存储空间　　D．所需空间与线性表的长度正比

2．假设有10个关键字，它们具有相同的Hash函数值，用线性探测法把这10个关键字存入Hash地址空间中至少要做（　　）次探测。

A．110　　B．100　　C．55　　D．45

3．某二叉树的前序序列和中序序列正好相反，则该二叉树一定具有（　　）的特征。

A．二叉树为空或只有一个结点
B．若二叉树不为空，则任一结点不能同时拥有左孩子和右孩子
C．若二叉树不为空，则任一结点没有左孩子
D．若二叉树不为空，则任一结点没有右孩子

4．在有n个叶子结点的哈夫曼树中，其结点总数为（　　）。

A．n　　B．$2n$　　C．$2n+1$　　D．$2n-1$

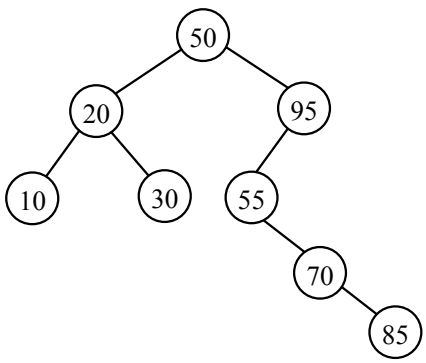
5．下列排序算法中，（　　）算法可能会出现下面情况：初始数据有序时，花费时间反而最多。

A．堆排序　　B．冒泡排序　　C．快速排序　　D．希尔排序

6．具有2000个结点的二叉树，其高度至少为（　　）。（注：空的二叉树的高度为-1，非空的二叉树的高度为其左、右子树的高度的较大者加1）

A．9　　B．10　　C．11　　D．12　　E．13

7．一棵二叉树如下图，该树是（　　）。



A．平衡树　　B．查找树　　C．堆　　D．以上都不是

二、应用题（每小题8分，共24分）

1．对如图1所示的图，写出其邻接矩阵。画出用Kruskal算法构造其最小生成树的每步结果（只要求用图表示即可）。

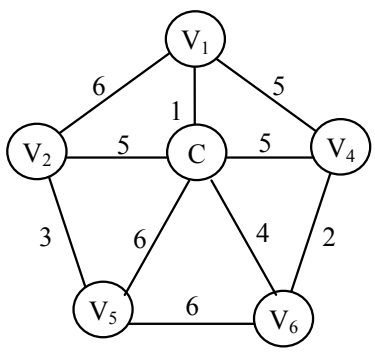


图 1

2．对一组关键字：26，5，37，1，62，11，59，15采用快速排序方法进行排序，用第一关键字作划分元素，请写出每趟划分的结果。

3．在C语言中用数组Q[n]以顺序存储的方法实现环形队列，头指针为f，指向队列的头结点所在的位置，尾指针为r，指向队列的尾结点后面的一个位置，请给出计算队列中的元素个数、判断队列是否为空的和判断队列是否为满的方法，画出你的方案的示意图。

三、算法设计题

1．设在一个整数数组中存储了一个堆，请编写一C函数，将一个新的元素插入此堆，要求算法的时间复杂度和空间复杂度最小，请给出你的算法的时间复杂度和空间复杂度。（14分）

2．设有以标准形式存储的二叉树T，请编写两个C函数，分别计算T的深度和宽度。（注：空的二叉树的深度为-1，非空的二叉树的深度为其左、右子树的深度的较大者加1；空的二叉树的宽度为0，对一棵非空的二叉树各层所包含的结点个数分别进行计数，其中的最大者即为此非空的二叉树的宽度）。（8分+16分）

参 考 答 案

一、

1．A　　2．C　　3．D　　4．D

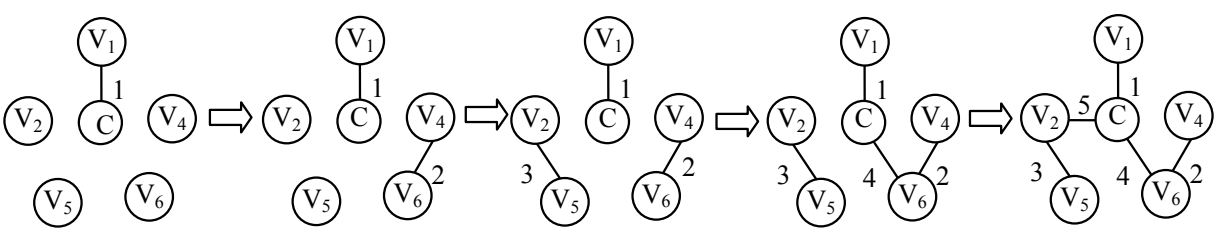
5．BC　　6．B　　7．B

二、

1．邻接矩阵为

	C	V_1	V_2	V_4	V_5	V_6
V_1	0	1	5	5	6	4
V_2	1	0	6	5	0	0
V_4	5	5	0	0	0	2
V_5	6	0	3	0	0	6
V_6	4	0	0	2	6	0

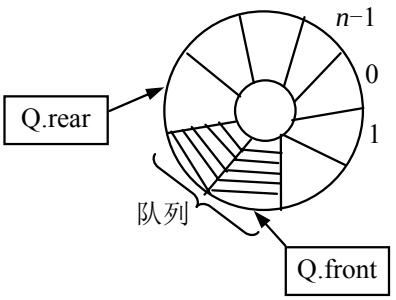
用Kruskal算法构造最小生成树的步骤如下



2．第一趟　15　5　11　1　26　62　59　37

　　第二趟　1　5　11　15　26　37　59　62

3．



牺牲一个元素空间，最多可存$n-1$元素

队列中元素的个数$n=(Q.front-Q.rear+n)\%n$;

当Q.front==Q.rear时，队列为空

当(Q.rear+1)%n==Q.front时，队列为满

三、

1．设堆的结构为

```
struct heap{
      int R[maxsize];   /*存放元素的空间*/
      int size;                  /*当前堆的元素的个数*/
}
```

算法为：

```
void heapInsert(heap.heap1;int x)
{     if(heap1.sizemaxsize)printf("overflow");
      else{
            heap1.size=leap1.size+1;
            while((i>1) && (heap1.R[parents(i)<x])){
                  heap1.R[i]=heap1.R[parents(i)];
                  i=parents(i);
            }
            heap1.R[i]=x;
      }
}
```

其中parent(i)是一函数，返回⌊i/2⌋

时间复杂度：$O(\log_2 n)$，其中n为堆中元素的个数

空间复杂度：$O(1)$

2．设树的结点为：

```
Struct BTreeNode{
      ElemType data;
      Struct BTreeNode *left;
      Struct BTreeNode *right;
}
```

求二叉树的深度的算法

```
int BTreeDepth(BTreeNode *T)
{
    if(T==NULL)return -1;
    else{
        int dep1=BTreeDepth(T->left);
        int dep2=BTreeDepth(T->right);
        if(dep1>dep2)return dep1+1;
        else return dep2+1;
    }
}
```

求二叉树的宽度的算法：写一个函数计算各层结点的个数

```
void total (BTreeNode *BT)
    { if(BT!=NULL){
        h=h+1;
        m=max(h，m);
        a[h]=a[h]+1;
        total(T->left);
        total(T->right);
    }
}
int BTreeWidth(BTreeNode *T)
{   int a[n];    //n要大于树的深度
    for(i=1;i<=n;i++) a[i]=0;
    int h=0;
    int m=0;
    total (T);
    x=a[1];
    for(i=2;i<=m;i++)
        if(x<a[i])x=a[i];
    return a[i];
}
```


北京邮电大学2003年硕士研究生入学考试试题

一、选择填空

1．5个圆盘的Hanoi塔，次小圆盘移到位时的步骤是第_____步。

A．16　　B．30　　C．31　　D．32

2．中缀表达式A*(B+C)/(D-E+F)的后缀表达式是_____。

A．A*B+C/D-E+F　　B．AB*C+D/E-F+

C．ABC+*DE-+/　　D．ABCDEF*+/-+

3．广义表$G=(a，b，(c，d，(e，f))，g)$的长度为_____。

A．3　　B．4　　C．7　　D．∞

4．对于有n个顶点e条边的连通图，其生成子图顶点和边的最小数目分别为_____和_____。

A．0　　B．n　　C．$n-1$　　D．e

5．含有4个元素值均不相同的结点的二叉排序树有_____种。

A．4　　B．6　　C．10　　D．14

6．有345个元素的有序表，等概率顺序查找成功的平均查找长度为_____。

A．86　　B．172　　C．173　　D．345

7．一棵m阶非空B-树，每个结点最多有_____棵子树；除根结点外，所有非终端结点最少有_____棵子树。

A．$\left\lceil \frac{m}{2} \right\rceil$　　B．$m-1$　　C．m　　D．$m+1$

8．就平均时间而言，下列排序方法中_____最好。

A．直接插入排序　　B．快速排序　　C．堆排序　　D．归并排序

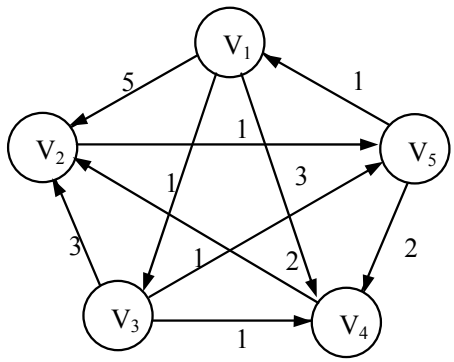
二、判断对错

1．数据的逻辑结构与数据元素本身的形式和内容无关。
2．线性表的逻辑顺序总与其物理顺序一致。
3．字符串‘ababaab’的改进的失败函数nextval的值是‘0101011’。
4．（10，21，43，39，22，45，48，201，49，46，99）是堆。
5．任何一个关键活动提前完成，则整个工程也会提前完成。
6．哈夫曼树的所有子树也均是哈夫曼树。
7．平衡二叉树（AVL树）的中序遍历值是递增的。
8．折半查找适用于所有的有序表。
9．理想情况下散列表等概率查找成功的平均查找长度是$O(1)$。
10．求n个数中最大的k(k<<n个)数，起泡排序比直接选择排序要好。

三、以下表顺序建立平衡二叉排序树，并求在等概率情况下查找成功的平均查找长度。

（90，60，20，50，40，30，10，110，100，70，120，80）

四、用迪杰斯特拉（Dijkstra）算法求下图中V_1顶点到其他各顶点的最短距离和最短路径，请写出求解过程。



五、算法

1．现有算法如下，求给定输入时的输出：

```
TYPE list= ↑ node;
        node=Record
                data:integer;
                next:list
             End;
PROC RDWRT;
  H:=nil;   Read(a);
  While a>0   Do
  Begin
      New(P);
      P ↑ .data:=a;
      P ↑ .next:=H;
      H:=P;
      Read(a)
 End;
 Q:=P ↑ .next;
 While Q≠nil Do
 Begin
   R:=Q ↑ .next;
   If R≠nil Then
   Begin
      Q ↑ .next:=R ↑ .next;
      R ↑ .next:=Q;
      P ↑ .next:=R;
      P:=Q;
      Q:=P ↑ .next;
    End
  Else Q:=nil
  End;
  P:=H;
  While p≠nil Do
  Begin
     Write(P ↑ .data，  ' ');
     P:=P ↑ .next;
  End
END;
```

输入为：1　2　3　4　5　6　7　8　9　10　0

2．已知以二叉链表表示的二叉树中有值为e、e1、e2的三个结点，下面的算法是判断e是否为e1和e2的共同祖先，请在空格处填上相应的语句或表达式。

```
TYPE biptr=↑.node;
        node=Record
            data:datatype;
            lc，rc:biptr
          End;
 FUNC Forefather(t:biptr; e，e1，e2:integer):boolean;
    f:=p1:=p2:=nil;
    Search(t，f，e);
    [    S1;    ]
    [    S2;    ]
    If [   S3   ] Then Return(true);
    Return(false)
ENDF;

PROC Search(t:biptr; var s:biptr;   e:datatype);
    If [   S4   ]  And   (s=nil) Then
    Begin
        If t↑.data=e   Then s:=t;
        [     S5;     ]
        Search(t↑.lc，s，e)
     End
ENDP;
```

3．求以二叉链表表示的三叉树中叶子结点的个数，在求值过程中，将树中所有结点的左右子树重新调换；结点值大的子树作为右子树，空结点值最小，试写出算法。

参 考 答 案

一、

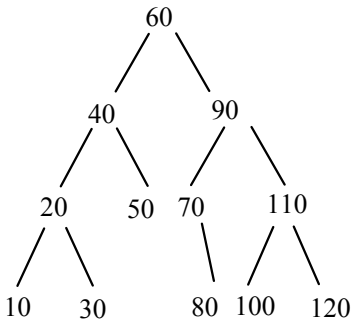
1. B	2. C	3. B	4. B，C
5. D	6. C	7. C，A	8. B

二、

1. ×	2. ×	3. √	4. √	5. √
6. √	7. √	8. √	9. √	10. √

三、

得到的平衡二叉排序树如下：



平均查找长度*ASL*=(1+2×2+3×4+4×5)/12=37/12

四、

终点	从V_1到各终点的dist值和最短路径			
	5	4	3	
V_2	(V_1，V_2)	(V_1，V_3，V_2)	(V_1，V_3，V_4，V_2)	
V_3	(V_1，V_3) 1			
	3	2		
V_4	(V_1，V_4)	(V_1，V_3，V_4)		
		3	3	3
V_5	∞	(V_1，V_3，V_5)	(V_1，V_3，V_5)	(V_1，V_3，V_5)
V_j	V_3	V_4	V_2	V_5

五、

1. 输出：10 8 9 6 7 4 5 2 3 1

2. S1：Search (f，p1，e1)
   S2：Search (f，p2，e2)
   S3：(p1<>nil) AND (p2<>nil)
   S4：(t<> nil)
   S5: Search(t↑.rc，s，e)

3. TYPE biptr= ↑node;
   node=Record
   data:datatype;
   lc，rc:biptr;
   End;

```
FUNC leafnum (t:biptr):integer;
  If (t↑.lc=nil) And (t↑.rc=nil) THEN Return(1);{叶子结点}
  If (t↑.lc<>nil) And (t↑.rc<>nil) THEN
  Begin
     If (t↑.lc.data>t↑.rc↑.data) THEN
     Begin
        temp:=t↑.lc;    t↑.lc:=t↑.rc;   t↑.rc:=temp
     End;
     Return(leafnum(t↑.lc)+leafnum(t↑.rc))
  End;
  If (t↑.lc<>nil) THEN
  Begin
       t↑.rc:=t↑.lc;
       t↑.lc:=nil;
```

```
    End;
    Return(leaf num(t ↑ .rc))
ENDF;
```

北京科技大学2003年硕士研究生入学考试试题

一、（30分）回答下列各题：

1．简述数据结构中线性结构、层次结构和网状结构的特点。
2．在算法正确的情况下，应从哪几个方面来衡量一个算法的优劣性？
3．设当前队列Q中有n（$n\geqslant 1$）个元素，即$Q=(a_1, a_2, \cdots, a_n)$，请画出队列$Q$的链式存储结构。
4．设二叉树中结点数为n（$n\geqslant 1$），树中第i（$1\leqslant i\leqslant n$）个结点的出度为$od(i)$，请写出n与树中各结点度之间的关系。
5．若一棵h层的完全二叉树中叶子结点数为n，且第h层的结点数$\geqslant 2$，则h=？
6．一个有向图在计算机存储器中的映像（或表示）通常有哪几种方法？
7．求一个有向无环图的拓扑序列时，其结果为何不惟一？
8．将n（$n\geqslant 1$）个结点组织成何种形态的二叉排序树时，对其查找时的时间复杂度为最佳？
9．在构造一个Hash表的过程中，简述如何用“链地址法”来解决冲突。
10．在“快捷排序”、“堆排序”和“归并排序”三个排序算法中，当只需获取前几个最小关键字记录时，选取何种排序算法为最佳？

二、（20分）算法填空：

建立在单链表上的一个C语言描述算法如下，其中L为链表结点的指针。请填充算法中下划线的空白之处，并简述算法完成的功能。

（注：答案请写在答卷纸上）

```
typedef struct node
{       int   data;
        struct node *next;
}       Lnode，*link;
void Selectsort(link L)
{       link p，q，minp;
        int temp;
        p=L->next;
        while(      (1)      )
       {     (2)      ;
          q=p->next;
          while(      (3)      )
          {  if(q->data<minp->data)
                      (4)      ;
             q=q->next;}
```

```
        if(_____（5）_____)
        {temp=p->data;
          p->data=minp->data;
          minp->data=temp;}
                _____（6）_____;
    }
}
```

三、（16分）设矩阵

$$A=\begin{bmatrix}0&0&1&0&3\\0&2&0&0&0\\0&0&0&1&5\\0&0&0&1&0\\0&0&0&0&8\end{bmatrix}$$（行列下标i、j满足：$1\leqslant i$，$j\leqslant 5$）

1．若将A视为一个上三角矩阵时，请画出对A进行“按行优先存储”的压缩存储表S，并写出A中元素之下标[i，j]与表S中元素之下标k（$1\leqslant k\leqslant 5$）之间的关系；

2．若将A视为一个稀疏矩阵时，请用C语言描述稀疏矩阵的三元组表，并画出A的三元组表结构。

四、（18分，此题统考生做）设记录的关键字集合：

$$K=\{12，3，14，30，6，17，7，28，2\}$$

1．请依次取K中各值，用插入的方法构造一棵3阶的B-树（插入的过程可省略，画出B-树的最后结构即可）；

2．简述B树结构有哪些基本的特点。

五、（18分）设考试科目为：英语（E）、数学（M）、程序设计（P）、数据结构（DS）、数据库（DB）、操作系统（OS），参加考试的学生报名表如下：

姓名	科目1	科目2	科目3
丁一	E	M	DS
王二	P	DB	OS
张三	E	DS	DB
李四	E	M	
赵五	DS	DB	OS

1．请根据报名表对考试时间安排的约束条件，构造以“考试科目”为数据元素集合的数据结构模型（提示：当某两个科目的考试不能同时进行时，将其连线，构造出的模型应是一种网状结构），用图G表示；

2．写出图G的邻接矩阵，画出图G的邻接表结构；

3．从图G中科目E出发，分别写出按“深度优先”和“广度优先”搜索方法遍历图G所得到的顶点序列。

六、（18分）设记录的关键字（key）集合：

K={15，25，26，4，5，20，3，12，2}

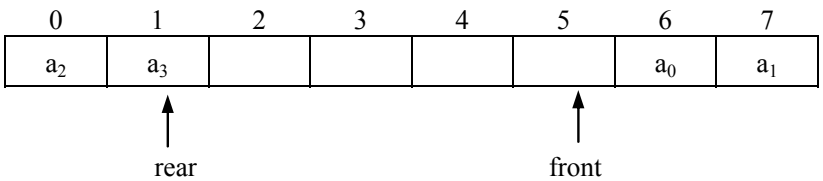
1. 以K为权值集合，构造一棵Huffman树，并回答如何利用遍历二叉树的算法求Huffman树的带权路径长度（WPL）。
2. 设Hash表表长m=16，选取Hash函数的方法为“除留余数法”，处理冲突的方法为“线性探测再散列”，请依次取K中各值，构造出满足所给事实上条件的Hash表；
3. 写出对K按“2路归并排序”方法排序时，各趟排序结束时的结果，并将K调整成一个堆顶元素取最小值的堆。

七、（30分，此题统考生做）算法设计：

1. 设二叉树采用链式结构存储，根结点指针为bt，结点的左、右子指针域分别为Lchild和Rchild，请采用中序非递归遍历二叉树的方法，用C语言函数形式写出求二叉树bt的最大层数（即深度）的算法：Btdep(bt)。
   （注：算法中可调用栈操作的基本函数）
2. 设有n个顶点的有向图G采用十字链表结构存储，请用C语言描述十字链表中的顶点、顶点表和弧结点，并写出求图G中第i(1≤i≤n)个顶点度的算法：Degree(G，i)。

（注：有向图中某个顶点的度定义为该顶点的入度与出度之和）

八、（18分，此题单考生做）设循环队列Q[8]的当前状态如下：



其中a_i(0≤i≤3)为队列Q中的元素，front和rear分别为队头和队尾指针（元素的下标）。

1. 写出队列Q的队空、队满判定条件及进队、出队操作的C语言描述语句；
2. 画出元素a_0，a_1，a_2，a_3出队、元素a_4，a_5，a_6，a_7进队后队列Q的状态。

九、（30分，此题单考生做）算法设计：

1. 设n个十进制整数已存入数组A[n]中，请利用栈技术，用C语言函数形式写出将A[n]中各数据转换成八进制数并存入数组B[n]的算法：Convert(A，B)。
   （注：算法中可调用栈操作的基本函数）
2. 设二叉树采用链式结构存储，根结点指针为bt，结点的左、右子指针域分别为Lchild和Rchild，请采用按层次遍历二叉树的方法，用C语言函数形式写出将二叉树中每一结点的左右子树相互交换的算法：Exchange(bt)。
   （注：算法中可调用队列操作的基本函数）

参 考 答 案

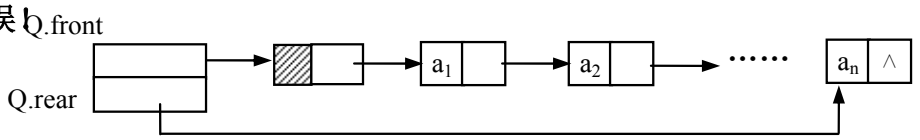
一、

1．线性结构：结构中数据元素是1个对1个。层次结构和网状结构：数据元素是多个对多个。

2．可读性、健壮性、效率与低存储量需求。

3．

错误！



4．$\sum_{i=1}^{n} od(i) = n-1$

5．$\lfloor \log_2 n \rfloor + 1$

6．数组表示法、邻接表、十字链表

7．因为一个结点可以到达多个不同的结点，所以在拓扑排序中可以有不同的结果。

8．平衡二叉树

9．将所有关键字为同义词的记录存储在同一线性链表中，假设某哈希数产生的哈希地址在区间[0，m−1]上，则设立一个指针型向量，chainhash：Array[0，m−1]of pointer；其中每个分量的初始状态都是空指针，凡哈希地址为i的记录都插入到头指针为chainhash[i]的链表中，在链表中的插入位置可以在表头或表尾，也可以在中间，以保持同义词在同一线性链表中按关键字有序。

10．堆排序

二、

（1）p!=NULL

（2）minp=p

（3）q!=NULL

（4）minp=q

（5）p->data>minp->data

（6）p=p->next

三、

1．$k = \dfrac{(10-i)(i-1)}{2} + j$，

存储表S	0	0	1	0	3	2	0	0	0	0	1	5	1	0	8
k=1	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

2．

```
#define MAXN 100      /*非零元素的最大值*/
struct tuple tp{      int i，j;   /*非零元所在的行，列号*/
                      int value; /*非零元的值*/
                };    /*三元组结构*/
struct sparmat {  int m，n，k;      /*稀疏矩阵的行数、列数及非零元个数*/
                  struct tuple 3tp data[MAXN];
                }; /*三元组表*/
```

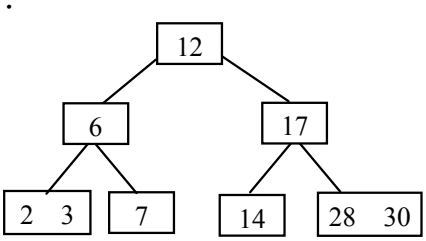
三元组表结构：

i	j	value
1	3	1
1	5	3
2	2	2
3	4	1
3	5	5
4	4	1
5	5	8

m=5，n=5，k=7

四、

1．



2．

B-树是一种平衡的多路查找树，一棵m阶的B-树，或为空树，或为满足下列特征的m叉树：

① 树中每个结点至多有m棵子树；

② 若根结点不是叶子结点，则至少有两棵子树；

③ 除根之外的所有非终端结点至少有$\lceil m/2 \rceil$棵子树；

④ 所有的非终端结点中包含下列信息数据：

（n，A_0，K_1，A_1，K_2，A_2，…，K_n，A_n）

其中，K_i(i=1，…，n)为关键字，且K_i<K_{i+1}，A_i为指向子树根结点的指针，且指针A_{i-1}所指子树中所有结点的关键字均小于K_i，A_n所指子树中所有结点的关键字均大

于K_n，n为关键字的个数。

⑤ 所有的叶子结点都出现在同一层次上，并且不带信息。

B+树是应文件系统所需而出的一种B-树的变型树，一棵m阶的B+树和m阶的B-树的差异在于：

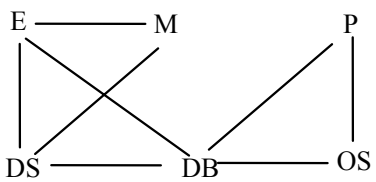
① 有n棵子树的结点中含有n个关键字；

② 所有的叶子结点中包含了全部关键字的信息及指向含这些关键字记录的指针，且叶子结点本身按关键字的大小自小而大顺序链接；

③ 所有的非终端结点可以看成是索引部分，结点中只含有其子树（根结点）中的最大（或最小）关键字。

五、

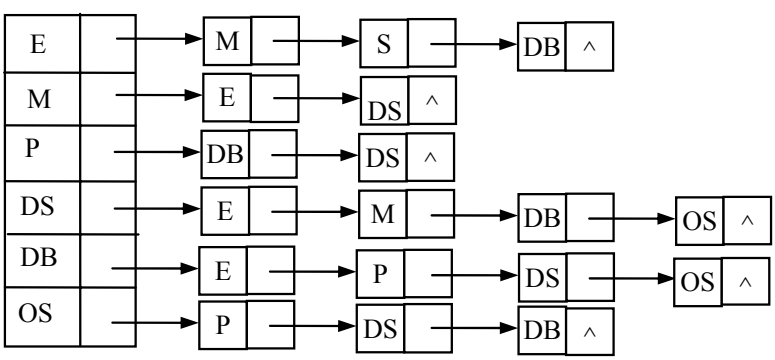
1.



2．邻接矩阵：

	E	M	P	DS	DB	OS
E	0	1	0	1	1	0
M	1	0	0	1	0	0
P	0	0	0	0	1	1
DS	1	1	0	0	1	1
DB	1	0	1	1	0	1
OS	0	0	1	1	1	0

邻接表：

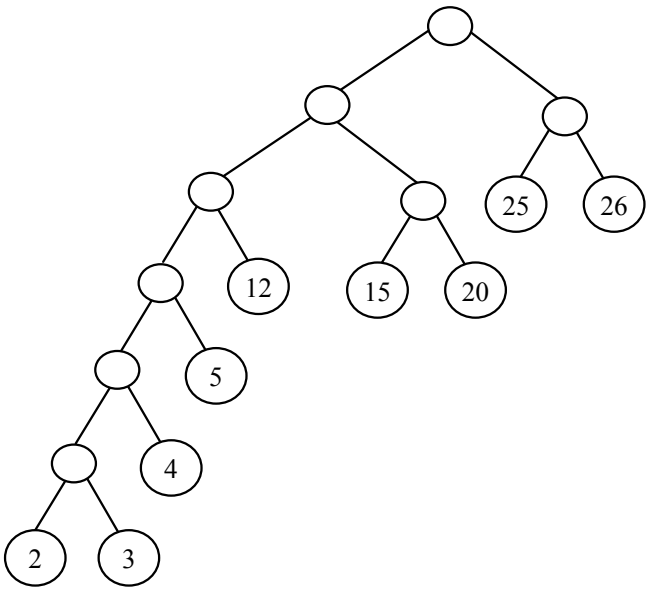


3．深度优先：E　M　DS　DB　OS　P

广度优先：E　M　DS　DB　OS　P

六、

1.

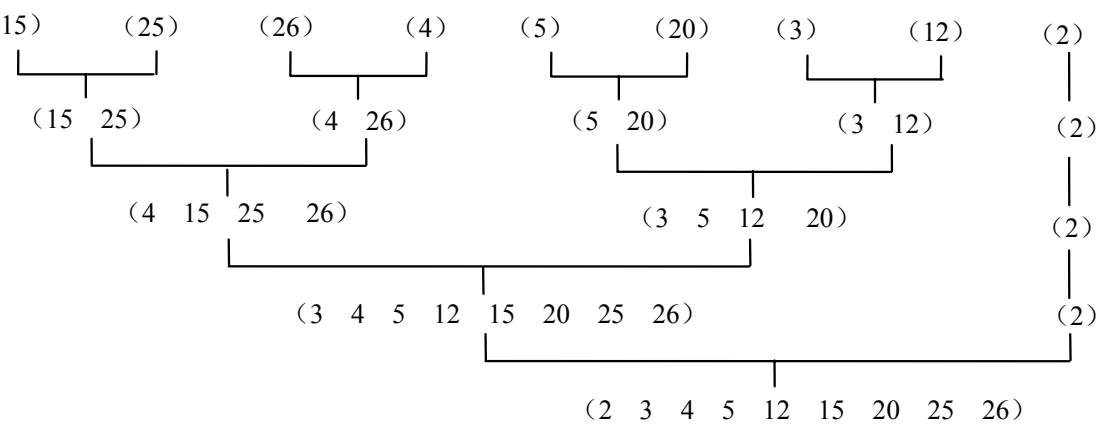


WPL=2×6+3×6+4×5+5×4+12×3+15×3+20×3+25×2+26×2=313

2．P取15

哈希表	15		2	3	4	5	20				25	26	12			
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

3.



错误!

七、

1.

```
int Btdep(NODE*bt)
{   int current dep，  maxdep;  //分别存放当前访问的层数与最大层数
    stack s1，s2;  //s1存放结点指针；s2中存放s1中对应位置结点的层数
    NODE *p;
    max dep=0;
    push(s1，bt);
    push(s2，1);
    while(! Empty Stack(s1))
    {   p=pop(s1);
        currentdep=pop(s2);
```

```
        if(currentdep>maxdep)   maxdep=currentdep;
        if (p->Lchild!=Null)
        {   push(s1，  p->Lchild)；
            push(s2，currentdep+1);
        }
        if(p->Rchild!=NULL)
        {   push(s1，p->Rchild);
            push(s2，curentdep+1);
        }
    }
  }
```

2．

```
#define MAXN 50      /*顶点数最大值*/
struct arcnode {  int tail vex，head vex;/*弧尾与弧头的顶点号*/
                  struct arcnode *hlink，*tlink;/*指向弧尾相同与弧头相同的下一条弧的指针*/
               }; /*弧结点*/
struct vexnode{   int vexnum;/*顶点编号*/
                  struct arcnode *firstin，*firstout;/*以该点为弧头、弧尾的第一个弧结点*/
               }; /*顶点*/
struct rextable {  int num;/*顶点数*/
                  struct rexnode data[MAXN];/*顶点数组*/
               };
int degree(struct vextable vt，int n)
{     struct arcnode *p;
      int i=0;
      p=vt.data[n]->first out;
      while(p!=NULL)
      {i++;p=p->tlink;}/*统计出度*/
      p=vt.data[n]->first in;
      while(p!=NULL)
      {i++;p=p->hlink};/*统计入度*/
      return i;
}
```

八、

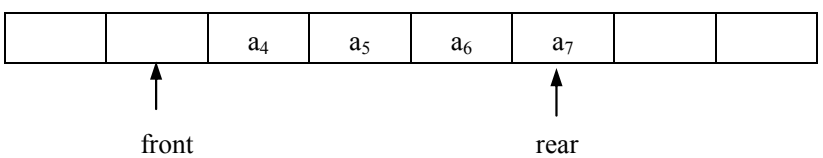
1．队空：rear=front

队满：(rear+1)%8=front

进队：Q[(++rear)%8]=data

出队：data=Q[(++front)%8]

2．



九、

1.

```
Convert (int A[ ]，int B[ ]，int n)
{    int i，p;
     for(i=0;i<n;i++)
     {    p=A[i];
          clearstack (s);
          while(p!=0)
          {    push(s，p%8);p/=8;}
               p=0;
               while(!Emptystack(s))
               p=p*10+pop(s);
               B[i]=p;
     }
}
```

2.

```
Exchange (struct node *bt)
{    struct node *p，*temp;
     clearqueue(Q);Enqueue(Q，bt);
     while(!EmptyQueue(Q))
     {    p=Dequeue(Q);temp=p->Lchild;p->Lchild=p->Rchild;p->Rchild=temp;
          if(p->Lchild!=NULL)Enqueue(Q，p->Lchild);
          if(p->Rchild!=NULL)Enqueue(Q，p->Rchild);
     }
}
```