第五章 数组和广义表

一、选择题

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1.B | 2.1L | 2.2J | 2.3C | 2.4I | 2.5C | 3.B | 4.B | 5.A | 6.1H | 6.2C | 6.3E |
| 6.4A | 6.5F | 7.B | 8.1E | 8.2A | 8.3B | 9.B | 10.B | 11.B | 12.B | 13.A | 14.B |
| 15.B | 16.A | 17.C | 18.D | 19.C | 20.D | 21.F | 22.C | 23.D | 24.C | 25.A | 26.C |
| 27.A |  |  |  |  |  |  |  |  |  |  |  |

二、判断题

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1. × | 2.√ | 3.√ | 4.× | 5.× | 6. × | 7.√ | 8.× | 9.× | 10.× | 11.× | 12.√ |
| 13.√ | 14.√ |  |  |  |  |  |  |  |  |  |  |

部分答案解释如下。

1. 错误。对于完全二叉树，用一维数组作存储结构是效率高的（存储密度大）。

4. 错误。数组是具有相同性质的数据元素的集合，数据元素不仅有值，还有下标。因此，可以说数祖是元素值和下标构成的偶对的有穷集合。

5. 错误。数组在维数和界偶确定后，其元素个数已经确定，不能进行插入和删除运算。

6. 错误。稀疏矩阵转置后，除行列下标及行列数互换外，还必须确定该元素转置后在新三元组中的位置。

8. 错误。广义表的取表尾运算，是非空广义表除去表头元素，剩余元素组成的表，不可能是原子。

9. 错误。广义表的表头就是广义表的第一个元素。只有非空广义表才能取表头。

10. 错误。广义表中元素可以是原子，也可以是表（包括空表和非空表）。

11. 错误。广义表的表尾，指去掉表头元素后，剩余元素所组成的表。

三、填空题

1. 顺序存储结构 2.（1）9572（2）1228 3.（1）9174（2）8788 4. 1100

5. 1164 公式：LOC(aijk)=LOC(a000)+[v2\*v3\*(i-c1)+v3\*(j-c2)+(k-c3)]\*l (l为每个元素所占单元数)

6. 232 7. 1340 8. 1196 9. 第1行第3列

10. (1)270 (2)27 (3)2204 11. i(i-1)/2+j (1<=i,j<=n)

12. (1)n(n+1)/2 (2)i(i+1)/2 (或j(j+1)/2) (3)i(i-1)/2+j (4)j(j-1)/2+i (1<=i,j<=n)

13. 1038 三对角矩阵按行存储：k=2(i-1)+j (1<=i,j<=n)

14. 33 (k=i(i-1)/2+j) (1<=i,j<=n)

15. 非零元很少(t<<m\*n)且分布没有规律 16. 节省存储空间。

17. 上三角矩阵中，主对角线上第r(1≤r≤n) 行有n-r+1个元素，aij所在行的元素数是j-i+1。所以，元素在一维数组的下标k和二维数组下标关系:k=((i-1)\*(2n-i+2))/2+(j-i+1)=(i-1)(2n-i)/2+j (i≤j)

18. 93 19. i(i-1)/2+j 20. 线性表 21. 其余元素组成的表

22. （1） 原子（单元素）是结构上不可再分的，可以是一个数或一个结构；而表带结构，本质就是广义表，因作为广义表的元素故称为子表。

（2）大写字母 （3）小写字母 （4）表中元素的个数（5）表展开后所含括号的层数

23. 深度 24.（1）（） （2）（（）） （3）2 （4）2

25. head（head（tail（tail（head（tail（tail（A）））））））

26. 表展开后所含括号的层数 27.（1）5 （2）3

28. head(head(tail(LS))) 29. head(tail(tail(head(tail(head(A))))))

30. head(tail(head(tail(H)))) 31. (b) 32. (x,y,z) 33. (d,e)

34. GetHead(GetHead(GetTail(L)))

35. 本算法中，首先数组b中元素以逆置顺序放入d数组中，然后数组a和数组d的元素比较，将大者拷贝到数组c。第一个WHILE循环到数组a或数组d结尾，第二个和第三个WHILE语句只能执行其中的一个。

（1）b[m-i+1]（2）x:=a[i]（3）i:=i+1（4）x:=d[j]（5）j:=j+1 （6）k:=k+1（7）i<=l（8）j<=m

36.（1）(i==k) **return**（2）i+1（3）i-1（4）i!=k

本算法利用快速排序思想，找到第k个元素的位置（下标k-1因而开初有k--）。内层do循环以t(t=a[low])为“枢轴”找到其应在i位置。这时若i等于k，则算法结束。（即第一个空格处if(i==k) **return**)。否则，若i<k，就在i+1至high中间去查；若i>k，则在low到i-1间去找，直到找到i=k为止。

37.逆置广义表的递归模型如下

f(LS)=

上面app**END**(a,b)功能是将广义表a和b作为元素的广义表连接起来。

（1）(p->tag==0) //处理原子

（2）h=reverse(p->val.ptr.hp) //处理表头

（3）(p->val.ptr.tp) //产生表尾的逆置广义表

（4）s->val.ptr.tp=t; //连接

（5）q->val.ptr.hp=h //头结点指向广义表

38. 本题要求将1，2，...,n\*n个自然数，按蛇型方式存放在二位数组A[n][n]中。“蛇型”方式，即是按“副对角线”平行的各对角线，从左下到右上，再从右上到左下，存放n2个整数。对角线共2n-1条，在副对角线上方的对角线，题目中用k表示第k条对角线（最左上角k=1），数组元素x和y方向坐标之和为k+1（即题目中的i+j=k+1）。副对角线下方第k条对角线与第2n-k条对角线对称，其元素的下标等于其对称元素的相应坐标各加(k-n)。

（1）k<=2\*n-1 //共填2\*n-1条对角线

（2）q=2\*n-k //副对角线以下的各条对角线上的元素数

（3）k%2！=0 //k为偶数时从右上到左下，否则从左下向右上填数。（本处计算下标i和j）

（4）k>=n //修改副对角线下方的下标i和j。

（5）m++；或m=m+1 //为填下个数作准备，m变化范围1..n\*n。

本题解法的另一种思路见本章算法设计题第9题。

39.本题难点有二：一是如何求下一出圈人的位置，二是某人出圈后对该人的位置如何处理。

按题中要求，从第s个人开始报数，报到第m个人，此人出圈。n个人围成一圈，可看作环状，则下个出圈人，其位置是(s+m-1)%n。n是人数，是个变量，出圈一人减1，算法中用i表示。对第二个问题，算法中用出圈人后面人的位置依次前移，并把出圈人的位置（下标）存放到当时最后一个人的位置（下标）。算法最后打印出圈人的顺序。

（1）(s+m-1) MOD i //计算出圈人s1

（2）s1:=i //若s1=0,说明是第i个人出圈（i%i=0）

（3）s1 TO i-1 //从s1到i依次前移，使人数减1，并将出圈人放到当前最后一个位置A[i]=w。

40. 若第n件物品能放入背包，则问题变为能否再从n-1件物品中选出若干件放入背包（这时背包可放入物品的重量变为s-w[n]）。若第n件物品不能放入背包，则考虑从n-1件物品选若干件放入背包（这时背包可放入物品仍为s）。若最终s=0,则有一解；否则，若s<0或虽然s>0但物品数n<1,则无解。

（1）s-w[n],n-1 //Knap(s-w[n],n-1)=true

（2）s,n-1 // Knap←Knap(s,n-1)

四、应用题

1、958 三维数组以行为主序存储，其元素地址公式为：

LOC(Aijk)=LOC(Ac1c2c3)+[(i-c1)V2V3+(j-c2)V3+(k-c3)]\*L+1

其中ci,di是各维的下界和上界，Vi=di-ci+1是各维元素个数，L是一个元素所占的存储单元数。

2. b对角矩阵的b条对角线，在主对角线上方和下方各有⎣b/2⎦条对角线(为叙述方便，下面设a=⎣b/2⎦)。从第1行至第a行，每行上的元素数依次是a+1,a+2,...,b-2,b-1,最后的a行上的元素个数是 b-1，b-2,...,a+1。中间的(n-2a )行，每行元素个数都是b。故b条对角线上元素个数为 (n-2a)b+a\*(a+b)。存放在一维数组V[1..nb-a(b-a)]中，其下标k与元素在二维数组中下标i和j的关系为：

k=

3．每个元素32个二进制位，主存字长16位，故每个元素占2个字长，行下标可平移至1到11。

（1）242 （2）22 （3）s+182 （4）s+142

4．1784 (公式：Loc(Aijkl)=100(基地址)+[(i-c1)v2v3v4+(j-c2)v3v4+(k-c3)v4+(l-c4)]\*4

5. 1210+108L (LOC(A[1,3,-2])=1210+[(k-c3)v2v1+(j-c2)v1+(i-c1)]\*L（设每个元素占L个存储单元）

6. 数组占的存储字节数=10\*9\*7\*4=2520；A[5,0,7]的存储地址=100+[4\*9\*7+2\*7+5]\*4=1184

7. 五对角矩阵按行存储，元素在一维数组中下标（从1开始）k与i,j的关系如下:

k=

A[15,16]是第71个元素，在向量[-10:m]中的存储位置是60 。

8．（1）540 （2）108 （3）i=3,j=10，即A[3,10] 9． k=i(i-1)/2+j

10. 稀疏矩阵A有t个非零元素，加上行数mu、列数nu和非零元素个数tu(也算一个三元组)，共占用三元组表LTMA的3(t+1)个存储单元，用二维数组存储时占用m\*n个单元，只有当3(t+1)<m\*n时，用LTMA表示A才有意义。解不等式得t<m\*n/3-1。

11．参见10。

12. 题中矩阵非零元素用三元组表存储，查找某非零元素时，按常规要从第一个元素开始查找，属于顺序查找，时间复杂度为O(n)。若使查找时间得到改善，可以建立索引，将各行行号及各行第一个非零元素在数组B中的位置（下标）偶对放入一向量C中。若查找非零元素，可先在数组C中用折半查找到该非零元素的行号，并取出该行第一个非零元素在B中的位置，再到B中顺序（或折半）查找该元素，这时时间复杂度为O(logn)。

13．（1）176 （2）76和108 （3）28和116。

14．（1）k = 3(i-1) (主对角线左下角，即i=j+1)

k = 3(i-1)+1 (主对角线上，即i=j)

k = 3(i-1)+2 (主对角线上，即i=j-1)

由以上三式，得 k=2(i-1)+j (1≤i,j≤n; 1≤k≤3n-2)

（2）103\*103-(3\*103-2)

15. 稀疏矩阵A采用二维数组存储时，需要n\*n个存储单元，完成求Σaii(1<=i<=n)时，由于a[i][i]随机存取，速度快。但采用三元组表时，若非零元素个数为t，需3(t+1)个存储单元（第一个分量中存稀疏矩阵A的行数，列数和非零元素个数，以后t个分量存各非零元素的行值、列值、元素值），比二维数组节省存储单元；但在求Σaii(1<=i<=n)时，要扫描整个三元组表，以便找到行列值相等的非零元素求和，其时间性能比采用二维数组时差。

16. 特殊矩阵指值相同的元素或零元素在矩阵中的分布有一定规律，因此可以对非零元素分配单元（对值相同元素只分配一个单元），将非零元素存储在向量中，元素的下标i和j和该元素在向量中的下标有一定规律，可以用简单公式表示，仍具有随机存取功能。而稀疏矩阵是指非零元素和矩阵容量相比很小（t<<m\*n），且分布没有规律。用十字链表作存储结构自然失去了随机存取的功能。即使用三元组表的顺序存储结构，存取下标为i和j的元素时，要扫描三元组表，下标不同的元素，存取时间也不同，最好情况下存取时间为O(1)，最差情况下是O(n)，因此也失去了随机存取的功能。

17．一维数组属于特殊的顺序表，和有序表的差别主要在于有序表中元素按值排序（非递增或非递减），而一维数组中元素没有按元素值排列顺序的要求。

18．n(n+1)/2（压缩存储） 或n2（不采用压缩存储）

19．LOC（A[i,j]）=LOC（A[3，2]）+[（i-3）\*5+（j-2）]×2 （按行存放）

LOC（A[i,j]）=LOC（A[3，2]）+[（j-2）\*6+（i-3）]×2 （按列存放）

20．n阶下三角矩阵元素A[i][j]（1<=i,j<=n,i>=j）。第1列有n个元素，第j列有n-j+1个元素，第1列到第j-1列是等腰梯形，元素数为(n+(n-j+2)(j-1)/2，而aij在第j列上的位置是为i-j+1。所以n阶下三角矩阵A按列存储，其元素aij在一维数组B中的存储位置k与i和j的关系为：

k=(n+(n-(j-1)+1)(j-1)/2+(i-j+1)=(2n-j)(j-1)/2+i

21．三对角矩阵第一行和最后一行各有两个非零元素，其余每行均有三个非零元素，所以共有3n-2个元素。

（1）主对角线左下对角线上的元素下标间有i=j+1关系，k与i和j的关系为k=3(i-1);主对角线上元素下标间有关系i=j，k与i和j的关系为k=3(i-1)+1; 主对角线右上那条对角线上元素下标间有关系i=j-1，k与i和j的关系为k=3(i-1)+2。综合以上三等式，有k=2(i-1)+j (1<=i,j<=n, |i-j|<=1)

（2）i=k/3+1；（1≤k≤3n-2） // k/3取小于k/3的最大整数。下同

j=k-2(i-1)=k-2(k/3)=k%3+k/3

22．这是一个递归调用问题，运行结果为：DBHEAIFJCKGL

23.（1）FOR循环中，每次执行PerfectShuffle(A,N)和CompareExchange(A,N)的结果：

第1次：A[1..8]=[90,30,85,65,50,80,10,100]

A[1..8]=[30,90,65,85,50,80,10,100]

第2次：A[1..8]=[30,50,90,80,65,10,85,100]

A[1..8]=[30,50,80,90,10,65,85,100]

第3次：A[1..8]=[30,10,50,65,80,85,90,100]

A[1..8]=[10,30,50,65,80,85,90,100]

（2）Demo的功能是将数组A中元素按递增顺序排序。

（3）PerfectShuffle 中WHILE循环内是赋值语句，共2N次，WHILE外成组赋值语句，相当2N个简单赋值语句；CompareExchange中WHILE循环内是交换语句，最好情况下不发生交换，最差情况下发生N次交换，相当于3N个赋值语句；Demo中FOR循环循环次数log22N，故按赋值语句次数计算Demo的时间复杂度为：最好情况：O(4N\*log22N)≈O(Nlog(2\*N))；最差情况：O((4N+3N)\*log22N)≈O(Nlog(2\*N))。

24. 这是一个排序程序。运行后B数组存放A数组各数在排序后的位置。结果是：

A={121,22,323,212,636,939,828,424,55,262}

B={3,1,6,4,8,10,9,7,2,5}

C={22,55,121,212,262,323,424,639,828,939}

25.（1）c= （2）a=

26.（1）同上面26题（1）

（2）对c数组的赋值同所选择的下标i和j的次序（指外层循环用j内层用i）没有关系

（3）同上题26（2）

（4）对i,j下标取反序后，重复执行第（3）步，A数组所有元素均变为2。（在机器上验证，反复循环3次后，所有元素均变为2）

27．错误有以下几处：

（1）过程参数没有类型说明； （2）出错条件判断：缺少OR（i+k>last+1）；

（3）删除元素时FOR循环应正向，不应用反向**DOWNTO**； （4）count没定义；

低效体现在两处：

（1）删除k个元素时，不必一个一个元素前移，而应一次前移k个位置；

（2）last指针不应一次减1，而应最后一次减k。

正确的高效算法如下：

const m=64；

**TYPE** ARR=ARRAY[1..m] OF integer；

**PROCEDURE** delk（**VAR** A:ARR；**VAR** last:integer;i,k：integer）；

{从数组A[1..last]中删除第i个元素起的k个元素，m为A的上限}

**VAR** count：integer；

**BEGIN**

IF（i<0）**OR**（i>last）**OR**（k<0）OR（last>m）**OR**（i+k>last+1）

**THEN** write（’error’）

**ELSE**[**FOR** count：= i+k **TO** last **DO** A[count-k]：=A[count]；

last：=last-k；]

**END**；

28. 这是计数排序程序。

（a）c[i](1<=i<=n)中存放A数组中值为i的元素个数。

（b）c[i](1<=i<=n)中存放A数组中小于等于i的个数。

（c）B中存放排序结果，B[1..n]已经有序。

（d）算法中有4个并列for循环语句，算法的时间复杂度为O(n)。

29．上三角矩阵第一行有n个元素，第i-1行有n-（i-1）+1个元素，第一行到第i-1行是等腰梯形，而第i行上第j个元素（即aij）是第i行上第j-i+1个元素，故元素Aij在一维数组中的存储位置（下标k）为：

k=(n+(n-(i-1)+1))(i-1)/2+(j-i+1)=(2n-i+2)(i-1)/2+j-i+1

30．将上面29题的等式进一步整理为：

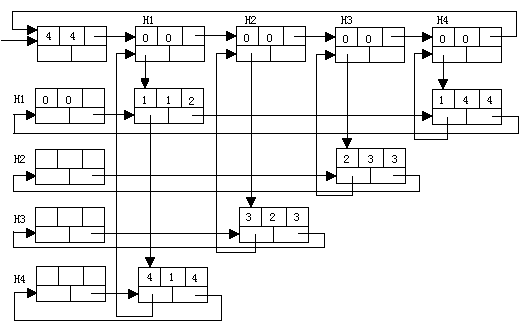
k=（n+1/2）i-i2/2+j-n，

则得f1（i）=（n+1/2）i-i2/2，f2（j）=j，c=-n。

31.（1）将对称矩阵对角线及以下元素按行序存入一维数组中，结果如下：



（2）因行列表头的“行列域”值用了0和0，下面十字链表中行和列下标均从1开始。



注：上侧列表头Hi和左侧行表头Hi是一个（即H1、H2、H3和H4），为了清楚，画成了两个。

32.（1）k=(2n-j+2)(j-1)/2+i-j+1 （当i≥j时，本题n=4）

k=(2n-i+2)(i-1)/2+j-i+1 （当i<j时，本题n=4）

（2）稀疏矩阵的三元组表为：s=((4,4,6),(1,1,1),(1,4,2),(2,2,3),(3,4,5),(4,1,2),(4,3,5)。其中第一个三元组是稀疏矩阵行数、列数和非零元素个数。其它三元组均为非零元素行值、列值和元素值。

33.（1）k=2（i-1）+j （1≤i，j≤n，|i-j|≤1）

i=floor（k/3）+1 // floor（a）是取小于等于a的最大整数

j=k-2（i-1）

推导过程见上面第25题。

（2）行逻辑链接顺序表是稀梳矩阵压缩存储的一种形式。为了随机存取任意一行的非零元，需要知道每一行第一个非零元在三元组表中的位置。为此，除非零元的三元组表外，还需要一个向量，其元素值是每行第一个非零元在三元组表中的位置。其类型定义如下：

**typedef struct**

{ **int** mu,nu,tu; //稀梳矩阵的行数、列数和非零元素个数

**int** rpos[maxrow+1]; //行向量，其元素值是每行第一个非零元在三元组表中的位置。

Triple data[maxsize];

}SparsMatrix;

因篇幅所限，不再画出行逻辑链接顺序表。

34．各维的元素数为di-ci+1，则a[i1，i2，i3]的地址为：

a0+[（i1-c1）（d3- c3+1）（d2- c2+1）+（i2-c2）（d2- c2+1）+（i3-c3）]\*L

35．主对角线上元素的坐标是i=j，副对角线上元素的坐标i和j有i+j=n+1的关系

（1）i=j或i=n+1-j （1≤i，j≤n）

（2）非零元素分布在两条主、副对角线上，除对角线相交处一个元素（下称“中心元素”）外，其余每行都有两个元素。主对角线上的元素，在向量B中存储的下标是k=2i-1(i=j, 1≤i，j≤n，1<=k<=2n-1)。

副对角线上的元素，在中心元素前，在向量B中存储的下标是k=2i(i<>j, 1≤i，j≤n/2)；在中心元素后，其下标是k=2(i-1)(i<>j，n/2+1≤i，j≤n, 1<=k<=2n-1)。

（3）aij在B中的位置K=

36. 由于对称矩阵采用压缩存储，上三角矩阵第一列一个元素，第二列两个元素，第j列j个元素。上三角矩阵共有n (n+1)/2个元素。我们将这些元素存储到一个向量B[n(n+1)/2+1]中。可以看到B[k]和矩阵中的元素aij之间存在着一一对应关系：



则其对应关系可表示为：k=( 1<=i,j<=n, 1<=k<=n(n+1)/2)

int MAX(int x,int y)

{ **return**(x>y?x:y);

}

int MIN(int x,int y)

{ **return**(x<y?x:y);

}

37. 设用mu,nu和tu表示稀疏矩阵行数，列数和非零元素个数，则转置矩阵的行数，列数和非零元素的个数分别是nu,mu和tu。转置可按转置矩阵的三元组表中的元素顺序进行，即按稀疏矩阵的列序，从第1列到第nu列，每列中按行值递增顺序，找出非零元素，逐个放入转置矩阵的三元组表中，转时行列值互换，元素值复制。按这种方法，第1列到第1个非零元素一定是转置后矩阵的三元组表中的第1个元素，第1列非零元素在第2列非零元素的前面。这种方法时间复杂度是O(n\*P)，其中p是非零元素个数，当p和m\*n同量级时，时间复杂度为O(n3)。

另一种转置方法称作快速转置，使时间复杂度降为O(m\*n)。它是按稀疏矩阵三元组表中元素的顺序进行。按顺序取出一个元素，放到转置矩阵三元组表的相应位置。这就要求出每列非零元素个数和每列第一个非零元素在转置矩阵三元组表中的位置，设置了两个附加向量。

38. 广义表中的元素，可以是原子，也可以是子表，即广义表是原子或子表的有限序列，满足线性结构的特性：在非空线性结构中，只有一个称为“第一个”的元素，只有一个成为“最后一个”的元素，第一元素有后继而没有前驱，最后一个元素有前驱而没有后继，其余每个元素有唯一前驱和唯一后继。从这个意义上说，广义表属于线性结构。

39. 数组是具有相同性质的数据元素的集合，同时每个元素又有唯一下标限定，可以说数组是值和下标偶对的有限集合。n维数组中的每个元素，处于n个关系之中，每个关系都是线性的，且n维数组可以看作其元素是n-1维数组的一个线性表。而广义表与线性表的关系，见上面38题的解释。

40．线性表中的元素可以是各种各样的，但必须具有相同性质，属于同一数据对象。广义表中的元素可以是原子，也可以是子表。其它请参见38

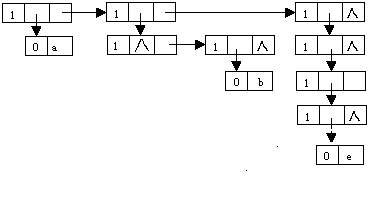
41.（1）（c,d） （2）（b） （3）b （4）（f） （5）（）

42. Head（Tail（Head（Head（L1））））

Head（Head（Head（Tail（Head（Tail（L2））））））

类似本题的另外叙述的几个题解答如下：

（1）head（head（tail（tail（L）））），设L=（a，（c），b），（（（e））））

（2）head（head（head（head（tail（tail（L））））））

（3）head（tail（head（tail（A））））

（4）H（H（T（H（T（H（T（L）））））））

（5）tail（L）=（（（c,d）），（e,f））

head（tail（L））=（（c,d））

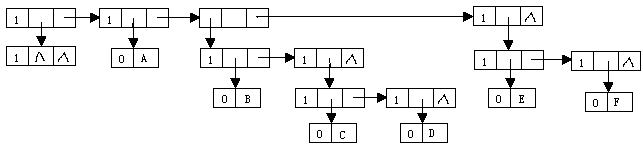
head（head（tail（L）））=（c,d）

tail（head（head（tail（L））））=（d）

head（tail（head（head（tail（L）））））=d

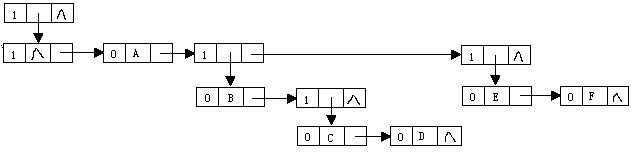
（6）head（tail（head（head（tail（tail（A））））））

43. 广义表的第一种存储结构的理论基础是，非空广义表可唯一分解成表头和表尾两部分，而由表头和表尾可唯一构成一个广义表。这种存储结构中，原子和表采用不同的结点结构（“异构”，即结点域个数不同）。



原子结点两个域：标志域tag=0表示原子结点，域DATA表示原子的值；子表结点三个域：tag=1表示子表，hp和tp分别是指向表头和表尾的指针。在画存储结构时，对非空广义表不断进行表头和表尾的分解，表头可以是原子，也可以是子表，而表尾一定是表（包括空表）。上面是本题的第一种存储结构图。

广义表的第二种存储结构的理论基础是，非空广义表最高层元素间具有逻辑关系：第一个元素无前驱有后继，最后一个元素无后继有前驱，其余元素有唯一前驱和唯一后继。有人将这种结构看作扩充线性结构。这种存储结构中，原子和表均采用三个域的结点结构（“同构”）。结点中都有一个指针域指向后继结点。原子结点中还包括标志域tag=0和原子值域DATA；子表结点还包括标志域tag=1和指向子表的指针hp。在画存储结构时，从左往右一个元素一个元素的画，直至最后一个元素。下面是本题的第二种存储结构图。



由于存储结构图占篇幅较大，下面这类题均不再解答。

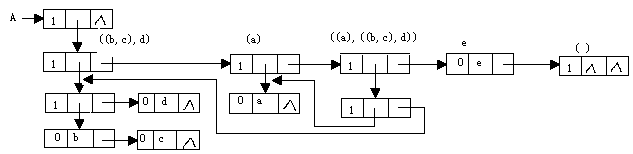
44.深度为5，长度为2

45.（1）略

（2）表的长度为5，深度为4

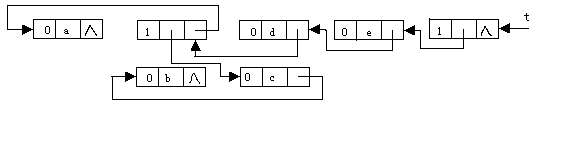
（3）head（tail（head（head（head（tail（tail（tail（tail（A）））））））））

46.共享结构广义表A=（（（b,c），d），（a），（（a），（（b,c），d）），e，（））的存储表示：



47.（1）算法A的功能是逆置广义表p（即广义表由p指针所指）。逆置后的广义表由t指向。

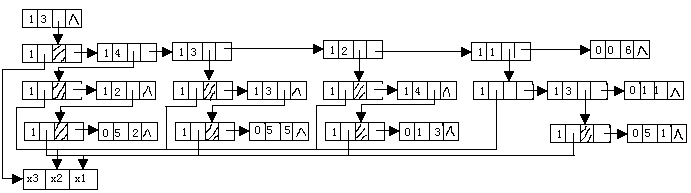
（2）逆置后的广义表由t指向，这时p=nil。



48.（a,b） 49.（d）

50.否。广义表的长度不是广义表中原子个数，而是指广义表中所含元素的个数，广义表中的元素可以是原子，也可以是子表。广义表元素多于1个时，元素间用逗号分开。

51. p（x1 x2 x3）=2 x15 x22 x34+5 x15 x23 x33+3 x1 x24 x32+（x15 x23+ x2）x3+6



52.（1）H(A(a1,a2）,B(b1）,C(c1，c2）,x）

HEAD(TAIL(HEAD(H）））=a2

（2）略

五.算法设计题

1.[题目分析]本题是在向量D内插入元素问题。首先要查找插入位置，数据x插入到第i个数据组的末尾，即是第i+1个数据组的开始，而第i（1≤i≤n）个数据组的首地址由数组s（即数组元素s[i]）给出。其次，数据x插入后，还要维护数组s，以保持空间区D和数组s的正确的相互关系。

**void** Insert（**int** s[]，datatype D[]，x，**int** i,m）

//在m个元素的D数据区的第i个数据组末尾，插入新数据x，第i个数据组的首址由数组s给出。

｛**if**（i<１|| i>n）｛printf（“参数错误”）；exit（０）；｝

**if**（i==n） D[m]=x； // 在第n个数据组末尾插入元素。

**else**｛**for**（j=m-1;j>=s[i+1];j--）D[j+1]=D[j]; // 第i＋１个数据组及以后元素后移

D[s[i+1]]=x; // 将新数据x插入

**for**(j=i+1;j<=n;j++) s[j]++; // 维护空间区D和数组s的的关系。

} //结束元素插入

m++; //空间区D的数据元素个数增1。

}// 算法Insert结束

[算法讨论] 数据在空间区从下标0开始，最后一个元素的下标是m-1。设空间区容量足够大，未考虑空间溢出问题。数组s随机存数，而向量D数据插入，引起数组元素移动，时间复杂度是O（n）。

2. [题目分析]设稀疏矩阵的非零元素的三元组以行序为主存储在三元组表中。矩阵的相加是对应元素的相加。对两非零元素相加，若行号不等，则行号大者是结果矩阵中的非零元素。若行号相同，则列号大者是结果中一非零元素；若行号列号相同，若对应元素值之和为零，不予存储，否则，作为新三元组存到三元组表中。题目中要求时间复杂度为O（m+n）。因此需从两个三元组表的最后一个元素开始相加。第一个非零元素放在A矩阵三元组表的第m+n位置上。结果的三元组至多是m+n个非零元素。最后若发生对应元素相加和为零的情况，对三元组表中元素要进行整理，以便使第一个三元组存放在下标1的位置上。

**CONST** maxnum=大于非零元素数的某个常量

**TYPE** tuple=**RECORD**

i,j：integer； v：elemtp；

**END**；

sparmattp=**RECORD**

mu，nu，tu：integer；

data: ARRAY[1..maxnum] OF tuple；

**END**；

**PROC** AddMatrix（**VAR** A：sparmattp；B：sparmattp）；

// 稀疏矩阵A和B各有m和n个非零元素，以三元组表存储。A的空间足够大，本算法实现两个稀疏矩阵相加，结果放到A中。

L:=m；p:=n；k:=m+n；// L,p为A，B三元组表指针，k为结果三元组表指针（下标）。

A.tu:=m+n；// 暂存结果矩阵非零元素个数

**WHILE**（L≥1）AND（p≥1）**DO**

[**CASE** // 行号不等时，行号大者的三元组为结果三元组表中一项。

A.data[L].i>B.data[p].i：A.data[k]:=A.data[L]；L:=L-1；// A中当前项为结果项

A.data[L].i<B.data[p].i：A.data[k]:=B.data[p]；p:=p-1；//B中当前项为结果当前项

A.data[L].i=B.data[p].i：

**CASE** //行号相等时，比较列号

A.data[L].j>B.data[p].j：A.data[k]:=A.data[L]；L:=L-1；

A.data[L].j<B.data[p].j：A.data[k]:=B.data[p]；p:=p-1；

A.data[L].j=B.data[p].j：**IF** A.data[L].v+B.data[p].v≠0 **THEN**

[A.data[L].v=A.data[L].v+ B.data[p].v；

A.data[k]:= A.data[L]；]

L:=L-1；p:=p-1；

**ENDC**； //结束行号相等时的处理

**ENDC**； //结束行号比较处理。

k:=k-1； //结果三元组表的指针前移（减1）

]//结束**WHILE**循环。

**WHILE** p>0 **DO**[A.data[k]:=B.data[p]；k:=k-1；p:=p-1；] //处理B的剩余部分。

**WHILE** L>1 **DO**[A.data[k]:=A.data[L]；k:=k-1；L:=L-1；] //处理A的剩余部分。

**IF** k>1 **THEN** //稀疏矩阵相应元素相加时，有和为零的元素，因而元素总数<m+n。

[**FOR** p:=k **TO** m+n **DO** A[p-k+1]:=A[p]；// 三元组前移，使第一个三元组的下标为1。

A.tu=m+n-k+1；] // 修改结果三元组表中非零元素个数。

**ENDP**； // 结束addmatrix

[算法讨论]算法中三元组的赋值是“成组赋值”，可用行值、列值和元素值的三个赋值句代替。A和B的三元组表的当前元素的指针L和p，在每种情况处理后均有修改，而结果三元组表的指针k在CASE语句后统一处理（k:=k-1）。算法在B的第一个元素“大于”A的最后一个元素时，时间复杂度最佳为O（n），最差情况是每个元素都移动（赋值）了一次，且出现了和为零的元素，致使最后（m+n-k+1）个元素向前平移一次，时间复杂度最差为O（m+n）。

3.[题目分析]从n个数中，取出所有k个数的所有组合。设数已存于数组A[1..n]中。为使结果唯一，可以分别求出包括A[n]和不包括A[n]的所有组合。即包括A[n]时，求出从A[1..n-1]中取出k-1个元素的所有组合，不包括A[n]时，求出从A[1..n-1]中取出k个元素的所有组合。

**CONST** n=10；k=3；

**TYPE** ARR=ARRAY[1..n] **OF** integer；

**VAR** A，B：ARR；// A中存放n个自然数，B中存放输出结果。

**PROC** outresult；//输出结果

**FOR** j：=1 **TO** k **DO** write（B[j]）；writeln；

**ENDP**；

**PROC** nkcombination（i,j,k：integer）；

//从i个数中连续取出k个数的所有组合，i个数已存入数组A中，j为结果数组B中的下标。

**IF** k=0 **THEN** outresult

**ELSE** **IF**（i-k≥0）**THEN** [ B[j]:=A[i]；j:=j+1；

nkcombination（i-1，k-1，j）；

nkcombination（i-1，k，j-1）；]

**ENDP**；

[算法讨论]本算法调用时，i是数的个数（题目中的n），k≤i，j是结果数组的下标。按题中例子，用nkcombination（5，1，3）调用。若想按正序输出，如123，124，…，可将条件表达式i-k≥0改为i+k-1≤n，其中n是数的个数，i初始调用时为1，两个调用语句中的i-1均改为i+1。

4. [题目分析]题目中要求矩阵两行元素的平均值按递增顺序排序，由于每行元素个数相等，按平均值排列与按每行元素之和排列是一个意思。所以应先求出各行元素之和，放入一维数组中，然后选择一种排序方法，对该数组进行排序，注意在排序时若有元素移动，则与之相应的行中各元素也必须做相应变动。

**void** Translation（float \*matrix，**int** n）

//本算法对n×n的矩阵matrix，通过行变换，使其各行元素的平均值按递增排列。

{**int** i,j,k,l；

float sum，min； //sum暂存各行元素之和

float \*p, \*pi, \*pk;

**for**(i=0; i<n; i++)

{sum=0.0; pk=matrix+i\*n; //pk指向矩阵各行第1个元素.

**for** (j=0; j<n; j++){sum+=\*(pk); pk++;} //求一行元素之和.

\*(p+i)=sum; //将一行元素之和存入一维数组.

}//for i

**for**(i=0; i<n-1; i++) //用选择法对数组p进行排序

{min=\*(p+i); k=i; //初始设第i行元素之和最小.

**for**(j=i+1;j<n;j++) **if**(p[j]<min) {k=j; min=p[j];} //记新的最小值及行号.

**if**(i!=k) //若最小行不是当前行,要进行交换(行元素及行元素之和)

{pk=matrix+n\*k; //pk指向第k行第1个元素.

pi=matrix+n\*i; //pi指向第i行第1个元素.

**for**(j=0;j<n;j++) //交换两行中对应元素.

{sum=\*(pk+j); \*(pk+j)=\*(pi+j); \*(pi+j)=sum;}

sum=p[i]; p[i]=p[k]; p[k]=sum; //交换一维数组中元素之和.

}//if

}//for i

free(p); //释放p数组.

}// Translation

[算法分析] 算法中使用选择法排序,比较次数较多,但数据交换(移动)较少.若用其它排序方法,虽可减少比较次数,但数据移动会增多.算法时间复杂度为O(n2).

5. [题目分析] 因为数组中存放的是从1到N的自然数，原程序运行后,数组元素A[i](1<=i<=N)中存放的是A[1]到A[i-1]中比原A[i]小的数据元素的个数。易见A[N]+1就是原A[N]的值(假定是j,1<=j<=N)。设一元素值为1的辅助数组flag，采用累加，确定一个值后，flag中相应元素置零。下面程序段将A还原成原来的A:

**VAR** flag:ARRAY[1..N] OF integer;

**FOR** i:=1 **TO** N **DO** flag[i]:=1; //赋初值

**FOR** i:=N **DOWNTO** 1 **DO**

**BEGIN** sum:=0; j:=1; found:=false;

**WHILE** j<=N AND NOT found **DO**

**BEGIN** sum:=sum+flag[j];

**IF** sum=A[i]+1 **THEN** **BEGIN** flag[j]:=0; found:=true; **END**;

**END**;

A[i]:=j;

**END**;

6.[题目分析] 寻找马鞍点最直接的方法,是在一行中找出一个最小值元素,然后检查该元素是否是元素所在列的最大元素,如是,则输出一个马鞍点,时间复杂度是O(m\*(m+n)).本算法使用两个辅助数组max和min,存放每列中最大值元素的行号和每行中最小值元素的列号,时间复杂度为O（m\*n+m），但比较次数比前种算法会增加，也多使用向量空间。

**int** m=10, n=10;

**void** Saddle(**int** A[m][n])

//A是m\*n的矩阵,本算法求矩阵A中的马鞍点.

{**int** max[n]={0}, //max数组存放各列最大值元素的行号,初始化为行号0;

min[m]={0}, //min数组存放各行最小值元素的列号,初始化为列号0;

i, j;

**for**(i=0;i<m;i++) //选各行最小值元素和各列最大值元素.

**for**(j=0;j<n;j++)

{**if**(A[max[j]][j]<A[i][j]) max[j]=i; //修改第j列最大元素的行号

**if**(A[i][min[i]]>A[i][j]) min[i]=j; //修改第i行最小元素的列号.

}

**for** (i=0;i<m;i++)

{j=min[i]; //第i行最小元素的列号

**if**(i==max[j])printf(“A[%d][%d]是马鞍点，元素值是%d”,i,j,A[i][j]); //是马鞍点

}

}// Saddle

[算法讨论] 以上算法假定每行（列）最多只有一个可能的马鞍点，若有多个马鞍点,因为一行(或一列)中可能的马鞍点数值是相同的，则可用二维数组min2，第一维是行向量，是各行行号，第二维是列向量，存放一行中最大值的列号。对最大值也同样处理，使用另一二维数组max2，第一维是列向量，是各列列号，第二维存该列最大值元素的行号。最后用类似上面方法，找出每行(i)最小值元素的每个列号（j），再到max2数组中找该列是否有最大值元素的行号（i），若有，则是马鞍点。

7．[题目分析]我们用l代表最长平台的长度，用k指示最长平台在数组b中的起始位置（下标）。用j记住局部平台的起始位置，用i指示扫描b数组的下标，i从0开始，依次和后续元素比较，若局部平台长度（i-j）大于l时，则修改最长平台的长度k（l=i-j）和其在b中的起始位置（k=j），直到b数组结束，l即为所求。

**void** Platform (**int** b[ ], **int** N)

//求具有N个元素的整型数组b中最长平台的长度。

{l=1;k=0;j=0;i=0;

**while**(i<n-1)

{**while**(i<n-1 && b[i]==b[i+1]) i++;

**if**(i-j+1>l) {l=i-j+1;k=j;} //局部最长平台

i++; j=i; } //新平台起点

printf(“最长平台长度%d，在b数组中起始下标为%d”，l，k)；

}// Platform

8．[题目分析]矩阵中元素按行和按列都已排序，要求查找时间复杂度为O（m+n），因此不能采用常规的二层循环的查找。可以先从右上角（i=a,j=d）元素与x比较，只有三种情况：一是A[i,j]>x， 这情况下向j 小的方向继续查找；二是A[i,j]<x，下步应向i大的方向查找；三是A[i,j]=x，查找成功。否则，若下标已超出范围，则查找失败。

void search(datatype A[ ][ ], **int** a,b,c,d, datatype x)

//n\*m矩阵A,行下标从a到b,列下标从c到d,本算法查找x是否在矩阵A中.

{i=a; j=d; flag=0; //flag是成功查到x的标志

**while**(i<=b && j>=c)

**if**(A[i][j]==x) {flag=1;break;}

**else** **if** (A[i][j]>x) j--; **else** i++;

**if**(flag) printf(“A[%d][%d]=%d”,i,j,x); //假定x为整型.

**else** printf(“矩阵A中无%d 元素”，x)；

}算法search结束。

[算法讨论]算法中查找x的路线从右上角开始，向下（当x>A[i,j]）或向左（当x<A[i,j]）。向下最多是m，向左最多是n。最佳情况是在右上角比较一次成功，最差是在左下角（A[b,c]），比较m+n次，故算法最差时间复杂度是O(m+n）。

9．[题目分析]本题的一种算法前面已讨论（请参见本章三、填空题38）。这里给出另一中解法。分析数的填法，是按“从右上到左下”的”蛇形”，沿平行于副对角线的各条对角线上，将自然数从小到大填写。当从右上到左下时，坐标i增加，坐标j减小，当j减到小于0时结束，然后j从0开始增加，而i从当前值开始减少，到i<0时结束。然后继续如此循环。当过副对角线后，在i>n-1时，j=j+2，开始从左下向右上填数；而当j>n-1时i=i+2，开始从右上向左下的填数，直到n\*n个数填完为止。

**void** Snake\_Number(**int** A[n][n],**int** n)

//将自然数1..n\*n,按”蛇形”填入n阶方阵A中.

{i=0; j=0; k=1; //i,j是矩阵元素的下标,k是要填入的自然数.

**while**(i<n && j<n)

{**while**(i<n && j>-1) //从右上向左下填数,

{A[i][j]=k++; i++ ;j--;}

**if**((j<0)&&(i<n)) j=0; //副对角线及以上部分的新i,j坐标.

**else** {j=j+2; i=n-1;} // 副对角线以下的新的i,j坐标.

**while**(i>-1 && j<n) //从左下向右上

{A[i][j]=k++; i--; j++;}

**if**(i<0 && j<n) i=0;

**else**{i=i+2; j=n-1;}

}//最外层**while**

}//Snake\_Number

10.[题目分析]判断二维数组中元素是否互不相同，只有逐个比较,找到一对相等的元素，就可结论为不是互不相同。如何达到每个元素同其它元素比较一次且只一次？在当前行，每个元素要同本行后面的元素比较一次（下面第一个循环控制变量p的for循环），然后同第i+1行及以后各行元素比较一次，这就是循环控制变量k和p的二层for循环。

**int** JudgEqual(ing a[m][n],**int** m,n)

//判断二维数组中所有元素是否互不相同，如是，返回1；否则，返回0。

{**for**(i=0;i<m;i++)

**for**(j=0;j<n-1;j++)

{ **for**(p=j+1;p<n;p++) //和同行其它元素比较

**if**(a[i][j]==a[i][p]) {printf(“no”); **return**(0); }

//只要有一个相同的，就结论不是互不相同

**for**(k=i+1;k<m;k++) //和第i+1行及以后元素比较

**for**(p=0;p<n;p++)

**if**(a[i][j]==a[k][p]) {printf(“no”); **return**(0); }

}// for(j=0;j<n-1;j++)

printf(yes”); **return**(1); //元素互不相同

}//算法JudgEqual结束

（2）二维数组中的每一个元素同其它元素都比较一次，数组中共m\*n个元素，第1个元素同其它m\*n-1个元素比较，第2个元素同其它m\*n-2 个元素比较，……，第m\*n-1个元素同最后一个元素(m\*n)比较一次,所以在元素互不相等时总的比较次数为 (m\*n-1)+(m\*n-2)+…+2+1=（m\*n）(m\*n-1)/2。在有相同元素时,可能第一次比较就相同,也可能最后一次比较时相同,设在(m\*n-1)个位置上均可能相同,这时的平均比较次数约为（m\*n）(m\*n-1)/4，总的时间复杂度是O(n4)。

11．二项式(a+b）n展开式的系数的递归定义为：

C(n,k)= C(n,k)=Cnk=

(1)**int** BiForm(**int** n,k) //二项式展开式的系数的递归算法

{**if**(n<0 || k<0 || k>=n) {printf(“参数错误\n ”);exit(0);}

**if**(k==0 || k==n) **return**(1);

**else** **return**(BiForm(n-1,k)+BiForm(n-1,k-1);

}

(2)C(6,4)的递归树

|  |
| --- |
| C(2,1)+C(2,0) |

|  |
| --- |
| C(6,4) |

|  |
| --- |
| C(2,2)+C(2,1) |

|  |
| --- |
| C(1,1)+C(1,0) |

|  |
| --- |
| C(1,1)+C(1,0) |

|  |
| --- |
| C(1,1)+C(1,0) |

|  |
| --- |
| C(2,2)+C(2,1) |

|  |
| --- |
| 1 |

|  |
| --- |
| 1 |

|  |
| --- |
| 1 |

|  |
| --- |
| 1 |

|  |
| --- |
| 1 |

|  |
| --- |
| 1 |

|  |
| --- |
| 1 |

|  |
| --- |
| 1 |

|  |
| --- |
| 1 |

|  |
| --- |
| 1 |

|  |
| --- |
| 1 |

|  |
| --- |
| 1 |

|  |
| --- |
| 1 |

|  |
| --- |
| C(1,1)+C(1,0) |

|  |
| --- |
| C(2,2)+C(2,1) |

|  |
| --- |
| C(3,3)+C(3,2) |

|  |
| --- |
| C(4,4)+C(4,3) |

|  |
| --- |
| 1 |

|  |
| --- |
| C(5,4) |

|  |
| --- |
| C(3,2) |

|  |
| --- |
| C(3,3)+C(3,2) |

|  |
| --- |
| C(3,1) |

|  |
| --- |
| C(4,3) |

|  |
| --- |
| C(4,2) |

|  |
| --- |
| C(5,3) |

|  |
| --- |
| + |

|  |
| --- |
| 15 |

|  |
| --- |
| 10 |

|  |
| --- |
| 6 |

|  |
| --- |
| 3 |

|  |
| --- |
| 3 |

|  |
| --- |
| 2 |

|  |
| --- |
| 2 |

|  |
| --- |
| 2 |

|  |
| --- |
| 2 |

|  |
| --- |
| 3 |

|  |
| --- |
| 4 |

|  |
| --- |
| 5 |

|  |
| --- |
| 3 |

|  |
| --- |
| 4 |

|  |
| --- |
| 1 |

|  |
| --- |
| + |

|  |
| --- |
| + |

(3)计算C(n,k)(0<=k<=n)的非递归算法

**int** cnk(**int** n,**int** k)

{**int** i; long x=1,y=1;

**for** (i=1;i<=k;i++) x\*=i;

**for** (i=n-k+1;i<=n;i++) y\*=i;

**return**(y/x)

}//cnk

12.[题目分析]本题属于排序问题，只是排出正负，不排出大小。可在数组首尾设两个指针i和j，i自小至大搜索到负数停止，j自大至小搜索到正数停止。然后i和j所指数据交换，继续以上过程，直到 i=j为止。

**void** Arrange(**int** A[],**int** n)

//n个整数存于数组A中，本算法将数组中所有正数排在所有负数的前面

{**int** i=0,j=n-1,x; //用类C编写，数组下标从0开始

**while**(i<j)

{**while**(i<j && A[i]>0) i++;

**while**(i<j && A[j]<0) j--;

**if**(i<j) {x=A[i]; A[i++]=A[j]; A[j--]=x; }//交换A[i] 与A[j]

}

}//算法Arrange结束.

[算法讨论]对数组中元素各比较一次，比较次数为n。最佳情况(已排好,正数在前,负数在后)不发生交换，最差情况(负数均在正数前面)发生n/2次交换。用类c编写，数组界偶是0..n-1。空间复杂度为O(1).

类似本题的其它题的解答:：

（1）与上面12题同，因要求空间复杂度也是O(n)，可另设一数组C，对A数组从左到右扫描，小于零的数在C中从左（低下标）到右（高下标）存，大于等于零的数在C中从右到左存。

（2）将12题中判定正数(A[i]>0)改为判偶数(A[i]%2==0)，将判负数(A[j]<0)改为(A[j]%2!=0)。

（3）同（2），只是要求奇数排在偶数之前。

（4）利用快速排序思想，进行一趟划分。

**int** Partition(**int** A[],**int** n)

//将n个元素的数组A调整为左右两部分，且左边所有元素小于右边所有元素，返回分界位置。

{**int** i=0,j=n-1,rp=A[0]; //设数组元素为整型

**while**(i<j)

{**while**(i<j &&A[j]>=rp) j--;

**while**(i<j &&A[i]<=rp) i++;

**if**(i<j) { x=A[i];A[i]=A[j]; A[j]=x; }

}

A[i]=rp; **return**(i); //分界元素

}// Partition

13. [题目分析] 设n个元素存放在数组A[1..n]中。设S初始为空集，可依次将数组A的每一个元素并入S，产生了含一个元素的若干集合，再以含一个元素的集合为初始集合，依次并入A的第二个（异于S的那个元素）元素并入S，形成了含两个元素的若干集合，……，如此下去，直至A[i]的全部元素并入。

**CONST** n=10;

**TYPE** datatype=char;

**VAR** A: array[1..n] OF datatype;

**PROC** powerset(s:set OF datatype)

[outset(s); //输出集合S

**FOR** i:=1 **TO** n **DO** powerset(S+A[i]);

]

**ENDP**;

调用本过程时，参数S为空集[]。

14、[题目分析]设稀疏矩阵是Amxn,Hm是总表头指针。设rch是行列表头指针，则rch->right=rch时该行无非零元素，用i记行号，用一维数组元素A[i]记第i行非零元个数。（为方便输出，设元素是整数。）

**int** MatrixNum(Olink Hm)

//输出由Hm指向的十字链表中每一行的非零元素个数

{Olink rch=Hm->uval.next**,** p;

**int** A[]; i=1;//数组A记各行非零元个数,i记行号

**while**(rch!=Hm)//循环完各行列表头

{p=rch->right; num=0; //p是稀疏矩阵行内工作指针,num记该行非零个数

**while**(p!=rch)//完成行内非零元的查找

{printf(“M[%d][%d]=%d”,p->row,p->col,p->uval.e);

num++;p=p->right; printf(“\n”);//指针后移 }

A[i++]=num;//存该行非零元个数

rch=rch->uval.next;//移到下一行列表头

}

num=0

**for**(j=1;j<i;j++)//输出各行非零元个数

{num+=A[j]; printf(“第%d行非零元个数为%d\n”,j,A[j]); }

**return**(num);//稀疏矩阵非零元个数

}算法结束

15、[题目分析] 广义表的元素有原子和表。在读入广义表“表达式”时，遇到左括号‘（’就递归的构造子表，否则若是原子，就建立原子结点；若读入逗号‘，’，就递归构造后续子表；若n=0，则构造含空格字符的空表，直到碰到输入结束符号（‘#’）。设广义表的形式定义如下：

**typedef** **struct** node

{**int** tag; //tag=0为原子，tag=1为子表

**struc**t node \*link; //指向后继结点的指针

**union** {**struct** node \*slink; //指向子表的指针

**char** data; //原子

}element;

}Glist;

Glist \*creat ()

//建立广义表的存储结构

{**char** ch; Glist \*gh;

scanf(“%c”,&ch);

**if**(ch==’’) gh=null;

**else** {gh=(Glist\*)malloc(sizeof(Glist));

**if**(ch==‘(’){gh->tag=1; //子表

gh->element.slink=creat(); } //递归构造子表

**else** {gh->tag=0;gh->element.data=ch;} //原子结点

}

scanf(“%c”,&ch);

**if**(gh!=null) **if**(ch==‘,’) gh->link=creat()； //递归构造后续广义表

**else** gh->link=null;

**return**(gh);

}

}算法结束

16、(1)略

(2)求广义表原子个数的递归模型如下

f(p)=

**PROC** Number(p:glist; **VAR** n: integer)

**VAR** m:integer;

n:=0;

**IF** p<>NIL **THEN**

[**IF** p^.tag=0 **THEN** n:=1 **ELSE** Number(p^.sublist,m)

n:=n+m; Number(p^.link,m); n:=n+m; ]

**ENDP**;

17. **int** Count(glist \*gl)

//求广义表原子结点数据域之和，原子结点数据域定义为整型

{**if**(gl==null) **return**(0);

**else** **if** (gl->tag==0) **return**((p->data)+count(gl->link));

**else** **return**(count(gl->sublist)+count(gl->link)); }

}// Count

18.（1）在n个正整数中，选出k(k<<m）个最大的数，应使用堆排序方法。对深度为h的堆，筛选算法中关键字的比较次数至多为2（h-1）次。建堆总共进行的关键字比较次数不超过4n，堆排序在最坏情况下的时间复杂度是O(nlogn）。

**int** r[1000]; // r[1000]是整型数组

（2）**void** sift(**int** r[],**int** k,m,tag)

//已知r[k+1..m]是堆，本算法将r[k..m]调整成堆，tag=1建立大根堆,tag=2建立小根堆

{i=k;j=2\*i;x=r[k];

**while** (j<=m)

{**if** (tag==2) //建立小根堆

{**if** (j<m && r[j]>r[j+1]) j++;//沿关键字小的方向筛选

**if**(r[j]<x)) {r[i]=r[j];i=j;j=2\*i;}

**else** break;}

**else** //建立大根堆

{**if** (j<m && r[j]<r[j+1]) j++;//沿关键字小的方向筛选

**if**(r[j]>x) {r[i]=r[j];i=j;j=2\*i;}

**else** break;}

}

r[i]=x;

}//sift

main(**int** argc,char \*argv[])

//根据命令行中的输入，从1000个数中选取n个最大数或n个最小数

{**int** m=1000,i,j;

n=augv[2]; //从命令行输入的第二个参数是需要输出的数的个数

**if**(n>m){printf(“参数错误\n”);exit(0);}

**for**(i=0;i<m;i++) scanf(“%d”,&r[i]); //输入1000个大小不同的正整数

**if** (augv[1]==‘a’) //输出n个最大数，要求建立大根堆

{**for**(i=m/2;i>0;i--) sift(r,i,m,1)

printf(“%d个最大数依次为\n”,n);

**for**(i=m;i>m-n+1;i--) //输出n个最大数

{printf(“%5d”,r[i]); j++; **if**((j+1)%5==0) printf(“\n”);//一行打印5个数

sift(r,1,i-1,1); } //调堆

}

**else** //(augv[1]==‘i’) //输出n个最小数，要求建立小根堆

{**for**(i=m/2;i>0;i--) sift(r,i,m,2)

printf(“%d个最小数依次为\n”,n);

**for**(i=m;i>m-n+1;i--) //输出n个最小数

{printf(“%5d”,r[i]); j++; **if**((j+1)%5==0) printf(“\n”);//一行打印5个数

sift(r,1,i-1,2); } //调堆

}

}//main

[算法讨论]算法讨论了建堆，并输出n（n小于等于m）个最大(小)数的情况，由于要求输出n个最大数或最小数，必须建立极大化堆和极小化堆。注意输出时的for循环控制到变量i从m变化到m-n+1，这是堆的性质决定的，只有堆顶元素才是最大(小)的。要避免使i从1到n来输出n个最大(小)数的错误。

19、[题目分析] 题目要求调整后第一数组（A）中所有数均不大于第二个数组（B）中所有数。因两数组分别有序，这里实际是要求第一数组的最后一个数A[m-1]不大于第二个数组的第一个数B[0]。由于要求将第二个数组的数插入到第一个数组中。因此比较A[m-1]和B[0]，如A[m-1]>B[0]，则交换。交换后仍保持A和B有序。重复以上步骤，直到A[m-1]<=B[0]为止。

**void** ReArranger (**int** A[],B[],m,n)

//A和B是各有m个和n个整数的非降序数组，本算法将B数组元素逐个插入到A中，使A中各元素均不大于B中各元素，且两数组仍保持非降序排列。

{ **while** (A[m-1]>B[0])

{x=A[m-1];A[m-1]=B[0]; //交换A[m-1]和B[0]

j=1;

wkile(j<n && B[j]<x) B[j-1]=B[j++]; //寻找A[m-1]的插入位置

B[j-1]=x;

x=A[m-1];i=m-2;

wkile(i>=0 && A[i]>x) A[i+1]=A[i--]; //寻找B[0]的插入位置

A[i+1]=x;

}

}算法结束

20、[题目分析]本题中数组A的相邻两段分别有序，要求将两段合并为一段有序。由于要求附加空间为O(1),所以将前段最后一个元素与后段第一个元素比较，若正序，则算法结束；若逆序则交换，并将前段的最后一个元素插入到后段中，使后段有序。重复以上过程直到正序为止。

**void** adjust(**int** A[],**int** n)

//数组A[n-2k+1..n-k]和[n-k+1..n]中元素分别升序，算法使A[n-2k+1..n]升序

{i=n-k;j=n-k+1;

**while**(A[i]>A[j])

{x=A[i]; A[i]=A[j]; //值小者左移，值大者暂存于x

k=j+1;

**while** (k<n && x>A[k]) A[k-1]=A[k++]; //调整后段有序

A[k-1]=x;

i--; j--; //修改前段最后元素和后段第一元素的指针

}

}算法结束

[算法讨论]最佳情况出现在数组第二段[n-k+1..n]中值最小元素A[n-k+1]大于等于第一段值最大元素A[n-k]，只比较一次无须交换。最差情况出现在第一段的最小值大于第二段的最大值，两段数据间发生了k次交换，而且每次段交换都在段内发生了平均（k-1）次交换，时间复杂度为O(n2)。

21、[题目分析]题目要求按B数组内容调整A数组中记录的次序，可以从i=1开始，检查是否B[i]=i。如是，则A[i]恰为正确位置，不需再调；否则，B[i]=k≠i，则将A[i]和A[k]对调，B[i]和B[k]对调，直到B[i]=i为止。

**void** CountSort (rectype A[],**int** B[])

//A是100个记录的数组，B是整型数组，本算法利用数组B对A进行计数排序

{**int** i,j,n=100;

i=1;

**while**(i<n)

{**if**(B[i]!=i) //若B[i]=i则A[i]正好在自己的位置上，则不需要调整

{ j=i;

**while** (B[j]!=i)

{ k=B[j]; B[j]=B[k]; B[k]=k; // B[j]和B[k]交换

r0=A[j];A[j]=A[k]; A[k]=r0; } //r0是数组A的元素类型,A[j]和A[k]交换

i++;} //完成了一个小循环，第i个已经安排好

}//算法结束

22、[题目分析]数组A和B的元素分别有序，欲将两数组合并到C数组，使C仍有序，应将A和B拷贝到C，只要注意A和B数组指针的使用，以及正确处理一数组读完数据后将另一数组余下元素复制到C中即可。

**void** union(**int** A[],B[],C[],m,n)

//整型数组A和B各有m和n个元素，前者递增有序，后者递减有序，本算法将A和B归并为递增有序的数组C。

{i=0; j=n-1; k=0;// i，j，k分别是数组A,B和C的下标，因用C描述，下标从0开始

**while**(i<m && j>=0)

**if**(a[i]<b[j]) c[k++]=a[i++] **else** c[k++]=b[j--];

**while**(i<m) c[k++]=a[i++];

**while**(j>=0) c[k++]=b[j--];

}算法结束

[算法讨论]若不允许另辟空间，而是利用A数组（空间足够大），则初始k=m+n-1，请参见第2章算法设计第4题。

23、[题目分析]本题要求建立有序的循环链表。从头到尾扫描数组A，取出A[i]（0<=i<n）,然后到链表中去查找值为A[i]的结点，若查找失败，则插入。

LinkedList creat(ElemType A[],**int** n)

//由含n个数据的数组A生成循环链表，要求链表有序并且无值重复结点

{LinkedList h;

h=(LinkedList)malloc(sizeof(LNode));//申请结点

h->next=h; //形成空循环链表

**for**(i=0;i<n;i++)

{pre=h;

p=h->next;

**while**(p!=h && p->data<A[i])

{pre=p; p=p->next;} //查找A[i]的插入位置

**if**(p==h || p->data!=A[i]) //重复数据不再输入

{s=(LinkedList)malloc(sizeof(LNode));

s->data=A[i]; pre->next=s; s->next=p;//将结点s链入链表中

}

}//for

**return**(h);

}算法结束