第一章

一、填空题

1 数据元素是数据的基本单位，..数据项.....是具有独立含义的最小标识单位。

3 数据之间的关系（逻辑结构）有四种 集合、线性结构、树形结构、网状结构或图状结构,可分为.....、.....两大类。

4 数据的存储结构包括..顺序存储结构.....、..链式存储结构.....

二、问答题

1. 什么是数据结构?什么是数据类型?

答: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操作对象以及他们之间的关系和操作等的学科。

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。

2. 叙述算法的定义与特性。

答：算法是对待定问题求解步骤的一种描述，他是指令的有限序列，其中每一条指令表示一个或多个操作。

一个算法具有以下 5 个重要特性：

1)、有穷性 2)、确定性 3)、 可行性 4)、输入 5)、输出

3. 叙述算法的时间复杂度。

答：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)，算法的时量度，记作 T（n）=O(f(n))

他表示随着问题规模 n 的增大，算法执行时间增长率和 f（n）的增长率相同，称作算法的渐进时间复杂度，简称时间复杂度。

三、判断题（在各题后填写“√”或“×”）

1. 线性结构只能用顺序结构来存放，非线性结构只能用非顺序结构来存放。（ × ）

2. 下列几种数量级从小到大的排列顺序为：

O(1) 、O(logn)、O(n) 、O(nlogn) 、$O(n^2)$ 、$O(n^3)$ 、$O(2^n)$ 。（ √ ）

四、1. 计算机执行下面的语句时，语句 s 的执行频度（重复执行的次数）为 _____ 。

```
FOR(i=1; i<n-1; i++)
 FOR(j=n;j>=i;j--)
 s;
```

2．有下列运行时间函数：

（1）$T_1(n)=1000$; （2）$T_2(n)=n^2+1000n$; （3）$T_3(n)=3n^3+100n^2+n+1$;

分别写出相应的大 O 表示的运算时间。（1）_____ （2）_____ （3）_____

3. 设 n 为正整数，利用大 O 记号，将该程序段的执行时间表示为 n 的函数，则下列程序段的时间复杂度可表示为 （1） ( O(n.) ) （2） ( $O(n^2)$ )

```
1)float sum1(int n){
      /* 计算 1！+2！+…+n！*/
      p=1; sum1=0;
      for (i=1; i<=n; ++i){
            p=p*i; sum1=sum1+p
            }
   }/* sum1 */

(2) float sum2(int n){
       /* 计算 1！+2！+…+n！*/
   sum2=0;
      for (i=1; i<=n; ++i){
          p=1;
          for (j=1; j<=i; ++j) p=p*j;
          sum2=sum2+p;}
    }/* sum2 */
```

第二章

一、判断

1. 线性表在顺序存储时，逻辑上相邻的元素未必在存储的物理位置次序上相邻。（ × ）

2. 顺序表结构适宜于进行顺序存取，而链表适宜于进行随机存取。（ × ）

二、填空

1．在单链表中，指针 p 所指结点为最后一个结点的条件是 p->next=NULL。

2．在单链的循环链表中，指针 p 所指结点为最后一个结点的条件是 p->next=头指针。

三、选择

1.、在一个长度为 n 的顺序表的表尾插入一个新元素的渐进时间复杂度为（ B ）

A．O (n)　　B．O (1)　　C．O ($n^2$)　　D．O ($\log_2 n$)

2.线性链表不具有的特点是（ A ）。

A．随机访问　　B．不必事先估计所需存储空间大小

C．插入与删除时不必移动元素　　D．所需空间与线性表长度成正比

3.线性表采用链式存储时，其地址（ D ）。

A 必须是连续的　　B 一定是不连续的

C 部分地址必须是连续的　　D 连续与否均可以.

4、下列哪一个程序片段是在链表中间插入一个结点。（假设新结点为 NEW，欲插入在 Pointer 结点之后）（ B ）

A NEW->next=Pointer　　B NEW->next=Pointer->next

Pointer=NEW　　Pointer->next=NEW

C Pointer->next=NEW->next　　D 以上皆非

NEW->next=Pointer

5. 在单链表中,增加头结点的目的是（ C ）　　A. 使单链表至少有一结点　　B. 标志表中首结点位置

C. 方便运算的实现　　D.说明单链表是线性表的链式存储实现

6． 线性表 L 在_____情况下适用于使用链式结构实现。（ B ）

（A）需经常修改 L 中的结点值　　（B）需不断对 L 进行删除插入

（C）L 中含有大量的结点　　（D）L 中结点结构复杂

7、向一个有 127 个元素原顺序表中插入一个新元素并保存原来顺序不变，平均要移动（ B ）个元素。

A、8　　B、63.5　　C、63　　D、7

三、算法设计

1 设顺序表 L 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。

```
1.  Status Insert_SqList(SqList &va,int x)//把 x 插入递增有序表 va 中
{if(va.length+1>va.listsize) return ERROR;
 va.length++;
 for(i=va.length-1;va.elem[i]>x&&i>=0;i--)
  va.elem[i+1]=va.elem[i];
 va.elem[i+1]=x;
 return OK;
}//Insert_SqList
```

2 分别写出算法将单链表和顺序表就地逆置（用尽可能少的附加空间在原存储出空间内将将线性表 a1,a2,a3,…an 逆置为 an…a3,a2,a1）。

2. 1

```
void reverse(SqList &A)//顺序表的就地逆置
{for(i=0,j=A.length-1;i<j;i++,j--)
   A.elem[i]<->A.elem[j];
}//reverse
```

2.2

```
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
```

分析:本算法的思想是,逐个地把 L 的当前元素 q 插入新的链表头部,p 为新表表头.

*3 删除元素递增排列的链表 L 中所有值相同的元素。

```
3.  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
```

第三章　作业

1.栈和队列都是特殊线性表，简述其相同点和差异处？

答：1)、相同点：栈和队列都是特殊线性表，对栈和队列的操作是对线性表操作的子集，他们是操作受限的线性表。

2)、不同点：栈是限定仅在表尾进行插入和删除操作的线性表；表尾端称为栈底，相应的表头端称为栈底：另外，栈的修改是按 LIFO 原则进行的。　　队列则是一种 FIFO 的线性表，它只允许在表的一端进行插入，而在表的另一端进行删除；允许插入的一端叫做队尾，允许删除的一端叫做队头。

2.在初始为空的队列中插入元素 a,b,c,d 以后，紧接着作了两次删除操作，此时的队尾元素是 [　d　]

*3.对于顺序存储的队列，存储空间大小为 n，头指针为 F，尾指针为 R。若在逻辑上看一个环，则队列中元素的个数为 [　（4）　]

(1).R-F　　⑵.n+R-F　　⑶.(R-F+1)mod n　　⑷.(n+R-F)mod n

4. 若一个序列的进栈顺序为 1,2,3,4, 那么不可能的出栈序列是　　[ A ]

A. 4,2,3,1　　B. 3,2,1,4　　C. 4,3,2,1　　D. 1,2,3,4

5、一个递归的定义可以用递归过程求解，也可以用非递归过程求解，但单从运行时间来看，通常递归过程比非递归过程（　B　）

A．较快　　B．较慢　　C．相同　　D 不确定

6.简述以下算法的功能（栈和队列的元素类型均为 int）

```
（1） Void alog 1 (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);  }      }
    //删除栈 S 中所有等于 e 的元素
（2）void proc_1(Stack S)
      { int i, n, A[255];
        n=0;
        while(!EmptyStack(S))
             {n++;  Pop(S,  A[n]);}
  for(i=1;  i<=n;  i++)
     Push(S,  A[i]);
   //将栈 S 逆序
（3） void alog3(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);  }
  }
  //将队列 Q 逆序
```

*7.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、入队和出队的算法。

```
解：  typedef Linklist CLQuene;
     int InitQueue (CLQueue *Q)
     int EnterQueue (CLQueue Q,QueueElementType X)
     int DeleteQuene (CLQueue Q,QueueElementType *X)
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  ERROR;//队列已空
p=Q->next->next;
x=p->data;
Q->next->next=p->next;
free(p);
return OK;
}//DeCiQueue
```

1．简述空串与空格串、主串与子串每对术语的区别？

答：1）、由一个或多个空格组成的串‘　’称为空格串（blank string）他的长度为串中空格字符的个数。　　不含任何元素符号的串称为空串，用符号“Φ”表示，他的长度为 0.

2）、串中任意个连续的字符组成的子序列称为该串的子串，包含子串的串相应的称为主串。

2．两个字符串相等的充要条件是什么？

答：两个串相等 <=> 两个串的值相等。

两个串的值相等，也即，两个串的长度相等并且各个对于位置的字符都相等。

3．串有哪几种存储结构？

答：1)、定长顺序存储结构；

2)、堆分配存储结构；

3)、串的块链存储结构。

4．已知两个串：s1=”fg cdb cabcadr”, s2=”abc”, 试求两个串的长度，判断串 s2 是否是串 s1 的子串，并指出串 s2 在串 s1 中的位置。

答：　s1 的长度：14；

s2 的长度：3；

串 s2 是串 s1 的子串，且 s2 在 s1 中的位置是 9.

5．已知：s1=〃I’m a student〃，s2=〃student〃，s3=〃teacher〃，试求下列各运算的结果：

Index(s1,s2,1)；SubString(sub,s,7,7);Strlength(s1)；Concat(s2,s3)；StrDelete (s1,4,10)；

答：运算结果依次是：7；Sub=“student”；13；　“student teacher”；　s1=“I’m”.

6.编写算法，统计串 S 中字符的种类和个数 （定长顺序存储）.

提示：用结构数组 T 存储统计结果

```
 typedef struct {
            char ch;
            int num;
           } mytype;
mytype T[MAXSIZE];
解:   Void strAnalyze(stringtype S)
     {   mytype T[MAXSIZE];
         for(i=1;i<=s[0];i++)
         {  c=s[i];  j=0;
            While(T[j].ch&&T[j].ch!=c)
            j++;
            if(T[j].ch)
              T[j].num++;
            else  T[j]={c,1}
          }
          for(j=0;T[j].ch;j++)
```

```
        printf("%c:%d\n",T[j].ch,T[j].num);
    } strAnalyze
```

## 一、选择题

5-1 二维数组 A[0..8,0..9]，其每个元素占 2 字节，从首地址 400 开始，按行优先顺序存放，则元素 A[8,5]的存储地址为__A____。

A) 570　　B) 506　　C) 410　　D) 482

5-2设有下三角矩阵A［0..10,0..10］，按行优先顺序存放其非零元素，每个非零元素占两个字节，存放的基地址为100，则元素A［5，5］的存放地址为（ D ）。

A) 110　　B) 120　　C) 130　　D) 14

5-3如下是一个稀疏矩阵的三元组法存储表示和基于此表示所得出的相关叙述

| 行下标 | 列下标 | 值 |
|---|---|---|
| 1 | 1 | 3 |
| 1 | 4 | 5 |
| 2 | 3 | 2 |
| 3 | 2 | 6 |
| 3 | 4 | 5 |
| 3 | 3 | 3 |

I.该稀疏矩阵有5行　　II.该稀疏矩阵有4列

III.该稀疏矩阵有6个非0元素

这些叙述中__C__是正确的。

A)仅I　　B)I和II　　C)仅III　　D)全部

5-4 按行优先顺序存储下三角矩阵的非零元素，则计算非零元素 $a_{ij}(1\le j\le i\le n)$的地址的公式为__D__。

A) $LOC(a_{ij})=LOC(a_{11})+i\times(i+1)/2+j$

B) $LOC(a_{ij})=LOC(a_{11})+i\times(i+1)/2+(j-1)$

C) $LOC(a_{ij})=LOC(a_{11})+i\times(i-1)/2+j$

D) $LOC(a_{ij})=LOC(a_{11})+i\times(i-1)/2+(j-1)$

5-5 对矩阵压缩存储是为了（ B ）。

A)方便　　B)节省空间　　C)方便存储　　D)提高运算速度

5-6 广义表((a),a)的表头和表尾分别是（ B ）。

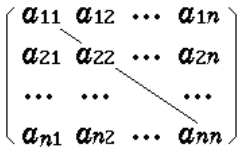
A) a ，((a))　　B) (a) ，(a)　　C) b，(a)　　D) ((a)) ， a

5-7 一个广义表为(a,(a,b),d,e,((i,j)k))，则该广义表的长度和深度分别是（ A ）。

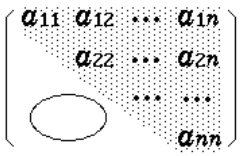
A) 5，3　　B)5，4　　C)4，3　　D)4，4

## 二、简答题

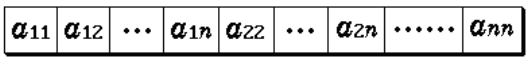
5-1 设有一个 $n\times n$ 的对称矩阵 $A$，如图(a)所示。为了节约存储，可以只存对角线及对角线以上的元素，称为上三角矩阵。我们把它们按行存放于一个一维数组 $B$ 中，如图(b)所示。并称之为对称矩阵 $A$ 的压缩存储方式。试问：若在一维数组 $B$ 中从 0 号位置开始存放，则如图(a)所示的对称矩阵中的任一元素 $a_{ij}$ 在只存上三角部分的情形下(图(b))应存于一维数组的什么下标位置？给出计算公式。



（a）对称矩阵



（b）只存上三角部分



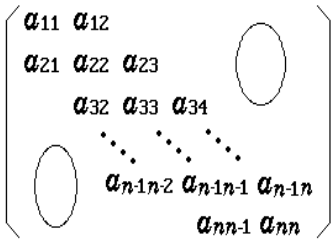
解：当 i＜j 或 i=j 时

LOC[ a（i，j）]=LOC[ a（0，0）] + [（2n-i-1）*i/2 +j]L

LOC[ a（i，j）]=LOC[ a（1，1）] + [（2n-i）*（i-1）/2 +j]L

当 i＞j 时 只需交换 i 和 j 即可。

5-2 在实际应用中经常遇到的稀疏矩阵是三对角矩阵，如图所示。在该矩阵中除主对角线及在主对角线上下最临近的两条对角线上的元素外，所有其它元素均为 0。现在要将三对角矩阵 $A$ 中三条对角线上的元素按行存放在一维数组 $B$ 中，且 $a_{11}$ 存放于 $B[0]$。试给出计算 $A$ 在三条对角线上的元素 $a_{ij}$（$1\le i\le n, i-1\le j\le i+1$)在一维数组 $B$ 中的存放位置的计算公式。




| a11 | a12 | a21 | a22 | a23 | a32 | a33 | a34 | …… | ann-1 | ann |
|---|---|---|---|---|---|---|---|---|---|---|

解：当 i=j 时　　a（i ， j）=3i 或 3j

当 i=j-1 时　a（i ， j）=3i+1 或 3j-2

当 i=j+1 时　a ( i　, j )=3i-1 或 3j+2

三、应用题

5-1 矩阵 $A_{m\times n}$ 中的某一元素 A[i][j]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的一个鞍点。假设以二维数组存放矩阵，试编写一个函数，确定鞍点在数组中的位置（若鞍点存在时），并分析该函数的时间复杂度.

解：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 ) 最小值是否鞍点

{ for(flag=1,k=0;k<m;k++)

if(min<A[k][j]) flag=0;

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

一、选择题

5-1 设有一个二维数组 *A* [ *m* ][ *n* ] ，假设 *A* [0][0] 存放位置在 644 (10) ， *A* [2][2] 存放位置在 676 (10) ，每个元素占一个空间，问 *A* [3][3] (10) 存放在什么位置（ ）脚注 (10) 表示用 10 进制表示。

A ） 676 B ） 650 C ） 600 D ） 692

【解析】

设数组元素 A[i][j] 存放在起始地址为 Loc ( i, j ) 的存储单元中。

∵ Loc ( 2, 2 ) = Loc ( 0, 0 ) + 2 * n + 2 = 644 + 2 * n + 2 = 676.

∴ n = ( 676 - 2 - 644 ) / 2 = 15

∴ Loc ( 3, 3 ) = Loc ( 0, 0 ) + 3 * 15 + 3 = 644 + 45 + 3 = 692.

【答案】 D

5-2 如下是一个稀疏矩阵的三元组法存储表示和基于此表示所得出的相关叙述

| 行下标 | 列下标 | 值 |
|---|---|---|
| 1 | 1 | 3 |
| 1 | 4 | 5 |
| 2 | 3 | 2 |
| 3 | 2 | 6 |
| 3 | 4 | 5 |
| 3 | 3 | 3 |

I. 该稀疏矩阵有 5 行 II. 该稀疏矩阵有 4 列

III. 该稀疏矩阵有 6 个非 0 元素

这些叙述中 _____ 是正确的。

A ） 仅 I B ） I 和 II C ） 仅 III D ）全部

【解析】本题考点是稀疏矩阵的三元组法存储。对稀疏矩阵的每一个非零元素，用一个三元组来描述。用线性表中的一个结点来对应稀疏矩阵的一个非零元素，每个结点包括三个域（行下标，列下标，值），结点之间的次序按矩阵的行优先顺列。

采用线性表的顺序存储结构来存储这些三元组，构成三元组表。

本题中，从该稀疏矩阵的三元组法存储表示中，只能判断该矩阵有 6 个非 0 元素，不能判断稀疏矩阵总的行数和列数。

【答案】 C

5-3 设有下三角矩阵 A［ 0..10,0..10 ］，按行优先顺序存放其非零元素，每个非零元素占两个字节，存放的基地址为 100 ，则元素 A［ 5 ， 5 ］的存放地址为（ ）。

A) 110 B) 120 C) 130 D) 140

【解析】 本题的考点是多维数组的顺序存储。按行优先顺序存储下三角矩阵 A nn 的非零元素，可以得到如下的序列： a 00 ,a 10 ,a 11 ,a 20 ,a 21 ,a 22, ….., a n0 ,a n1 ,a n3 , ….,a nn ，将该序列顺序存储在内存中，第 0 行到第 i-1 行的元素个数为 1 ＋ 2 ＋ … ＋（ i ） = i ′ (i+1)/2 ，假设 a 11 地址是 Loc(a 11 ) ，每个元素的长度为 k ，非零 a ij (1 ≤ j ≤ i ≤ n) 的是第 i 行的第 j 个元素，因此其地址是： Loc(a ij )= Loc(a 00 )+(i ′ (i+1)/2+j ）× k 。因此 A［ 5 ， 5 ］的存放地址 =100+ （ 5 × 6/2+5 ）× 2 ＝ 140 。

【答案】 D

5-4 按行优先顺序存储下三角矩阵的非零元素，则计算非零元素 a ij (1 ≤ j ≤ i ≤ n) 的地址的公式为 _____ 。

A) LOC(a ij )=LOC(a 11 ) +i×(i+1)/2+j

B) LOC(a ij )=LOC(a 11 )+i × (i+1)/2+ (j-1)

C) LOC(a ij )=LOC(a 11 ) +i×(i-1)/2+j

D) LOC(a ij )=LOC(a 11 )+i × (i-1)/2+ (j-1)

【解析】 本题的考点是多维数组的顺序存储。按行优先顺序存储下三角矩阵 A nn 的非零元素，可以得到如下的序列： a 11 ,a 21 ,a 22 ,a 31 ,a 32 ,a 33, ….., a n1 ,a n2 ,a n3 , ….,a nn ，将该序列顺序存储在内存中，第 1 行到第 i-1 行的元素个数为 1 ＋ 2 ＋ … ＋（ i-1 ） = i ′ (i-1)/2 ，假设 a 11 地址是 Loc(a 11 ), 非零 a ij (1 ≤ j ≤ i ≤ n) 的是第 i 行的第 j 个元素，因此其地址是： Loc(a ij )= Loc(a 11 )+ i ′ (i-1)/2+j-1 。

【答案】 D

5-5 二维数组 A[0..8,0..9] ，其每个元素占 2 字节，从首地址 400 开始，按行优先顺序存放，则元素 A[8,5] 的存储地址为 。

A) 570 B) 506 C ) 410 D) 482

【解析】 本题的考点是二维数组的存储。二维数组采用顺序存储结构，按行优先顺序，且下标从 0 开始，求数据元素的地址用下述公式： loc(a ij )=loc(a 11 )+ （ i*n+j ） * *l* ， 其中， n 和 m 分别为数组的每行和每列的元素个数， *l* 为每个数组元素所占用的存储空间单元个数。因此 A[8,5] 的地址是 400 ＋（ 8 × 10 ＋ 5 ）× 2 ＝ 570 。

【答案】 A

5-6 下面程序段的时间复杂度为 ( ).

```
Void mergesort(int i, int j)
{
int m;
if(i!=j){
m=(i+j)/2;
mergesort(i,m);
mergesort(m+1,j);
merge(i,j,m); }
}
```

A) O(n) B) O(n 2 ) C) O(nlogn) D) O(n!)

【解析】 其中 mergesort( ) 用于数组 a[n] 的归并排序，调用方式为 mergesort(0,n-1) ； merge( ) 用于两个有序子序列的合并，是非递归函数，它的时间复杂度为 O(n) 。

【答案】 A

二、简答题

5-1 设二维数组 A5*6 的每个元素占 4 个字节，已知 Loc （ a00 ） =1000 ， A 共占多少个字节？ A 的终端结点 a 45 的起始地址为多少？按行和按列优先存储时， a 25 的起始地址分别为多少？

【解析】 A 共占的字节数为： 5*6*4=120

a 45 的起始地址为： Loc （ a 00 ） + （ 4*6+5 ） *4 =1000+116=1116

按行优先存储时， a 25 的起始地址为： Loc （ a 00 ） + （ 2*6+5 ） *4=1000+68=1068

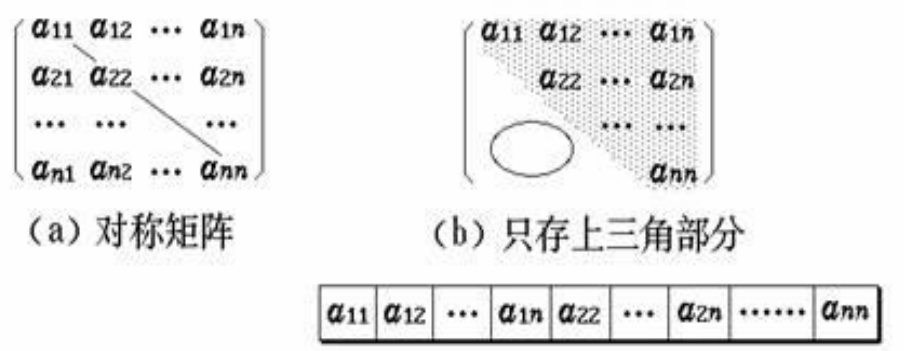
按列优先存储时， a 25 的起始地址为： Loc （ a 00 ） + （ 5*5+2 ） *4=1000+108=10108

5-2 设有一个 *n ′ n* 的对称矩阵 *A* 。为了节约存储，可以只存对角线及对角线以上的元素，或者只存对角线或对角线以下的元素。试问：存放对称矩阵 *A* 上三角部分或下三角部分的一维数组 *B* 有多少元素？

【解析】 数组 B 共有 n + ( n - 1 ) + ? ? ? ? ? ? + 1= n * ( n+1 ) / 2 个元素。

5-3 设有一个 *n ′ n* 的对称矩阵 *A* ，如图 (a) 所示。为了节约存储，可以只存对角线及对角线以上的元素，称为上三角矩阵。我们把它们按行存放于一个一维数组 *B* 中，如图 (b) 所示。并称之为对称矩阵 *A* 的压缩存

储方式。试问：若在一维数组 *B* 中从 0 号位置开始存放，则如图 (a) 所示的对称矩阵中的任一元素 *a ij* 在只存上三角部分的情形下 ( 图 (b)) 应存于一维数组的什么下标位置？给出计算公式。




【解析】 只存上三角部分时，若 i £ j ，则数组元素 A[i][j] 前面有 i - 1 行（ 1 ~ i - 1 ，第 0 行第 0 列不算 ），第 1 行有 n 个元素，第 2 行有 n - 1 个元素， ? ? ? ? ? ? ，第 i - 1 行有 n - i+2 个元素。在第 i 行中，从对角线算起，第 j 号元素排在第 j - i+1 个元素位置（从 1 开始），因此，数组元素 A[i][j] 在数组 B 中的存放位置为

n + (n - 1) + (n - 2) + ? ? ? ? ? ? ? + (n - i+2) + j - i+1

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

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

若 i > j ，数组元素 A[i][j] 在数组 B 中没有存放，可以找它的对称元素 A[j][i] 。在

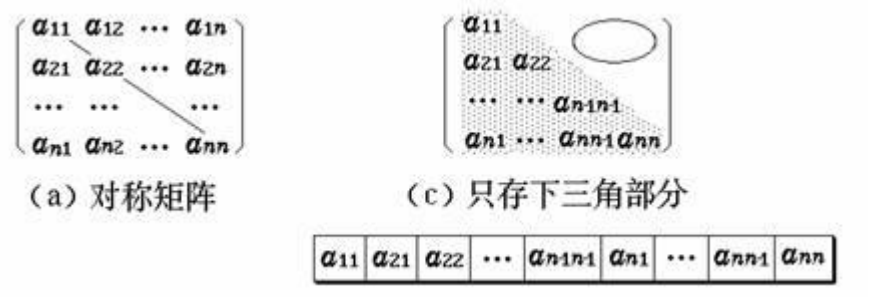
数组 B 的第 (2n - j) * (j - 1) / 2 + i 位置中找到。

如果第 0 行第 0 列也计入，数组 B 从 0 号位置开始存放，则数组元素 A[i][j] 在数组 B 中的存放位置可以改为

当 i £ j 时， = (2n - i+1) * i / 2 + j - i = ( 2n - i - 1 ) * i / 2 + j

当 i > j 时， = (2n - j - 1) * j / 2 + i

5-4 设有一个 *n* ′ *n* 的对称矩阵 *A* ，如图 (a) 所示。为了节约存储，可以只存对角线或对角线以下的元素，称为下三角矩阵。我们把它们按行存放于一个一维数组 *B* 中，如图 (c) 所示。并称之为对称矩阵 *A* 的压缩存储方式。试问： 若在一维数组 *B* 中从 0 号位置开始存放，则如图 (a) 所示的对称矩阵中的任一元素 *a ij* 在只存下三角部分的情形下 ( 图 (c)) 应存于一维数组的什么下标位置？给出计算公式。



【解析】只存下三角部分时，若 i 3 j ，则数组元素 A[i][j] 前面有 i - 1 行（ 1 ~ i - 1 ，第 0 行第 0 列不算 ），第 1 行有 1 个元素，第 2 行有 2 个元素， ? ? ? ? ? ? ，第 i - 1 行有 i - 1 个元素。在第 i 行中，第 j 号元素排在第 j 个元素位置，因此，数组元素 A[i][j] 在数组 B 中的存放位置为

1 + 2 + ? ? ? ? ? ? ? + (i - 1) + j = ( i - 1)*i / 2 + j

若 i < j ，数组元素 A[i][j] 在数组 B 中没有存放，可以找它的对称元素 A[j][i] 。在

数组 B 的第 (j - 1)*j / 2 + i 位置中找到。

如果第 0 行第 0 列也计入，数组 B 从 0 号位置开始存放，则数组元素 A[i][j] 在数组 B 中的存放位置可以改为

当 i 3 j 时， = i*(i+1) / 2 + j

当 i < j 时， = j*(j+1) / 2 + i

5-5 设有一个二维数组 *A* [ *m* ][ *n* ] ，假设 *A* [0][0] 存放位置在 644 (10) ， *A* [2][2] 存放位置在 676 (10) ，每个元素占一个空间，问 *A* [4][5] (10) 存放在什么位置？脚注 (10) 表示用 10 进制表示。

【解析】 设数组元素 A[i][j] 存放在起始地址为 Loc ( i, j ) 的存储单元中。

∵ Loc ( 2, 2 ) = Loc ( 0, 0 ) + 2 * n + 2 = 644 + 2 * n + 2 = 676.

∴ n = ( 676 - 2 - 644 ) / 2 = 15

∴ Loc ( 3, 3 ) = Loc ( 0, 0 ) + 4 * 15 + 5= 644 + 45 + 3 = 709

5-6 利用广义表的 head 和 tail 操作写出函数表达式，把以下各题中的单元素 banana 从广义表中分离出来：

(1) L1(apple, pear, banana, orange)

(2) L2((apple, pear), (banana, orange))

(3) L3(((apple), (pear), (banana), (orange)))

(4) L4((((apple))), ((pear)), (banana), orange)

(5) L5((((apple), pear), banana), orange)

(6) L6(apple, (pear, (banana), orange))

【解析】

(1) Head (Tail (Tail (L1) ) )

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

(3) Head (Head (Tail (Tail (Head (L3) ) ) ) )

(4) Head (Head (Tail (Tail (L4) ) ) )

(5) Head (Tail (Head(L5) ) )

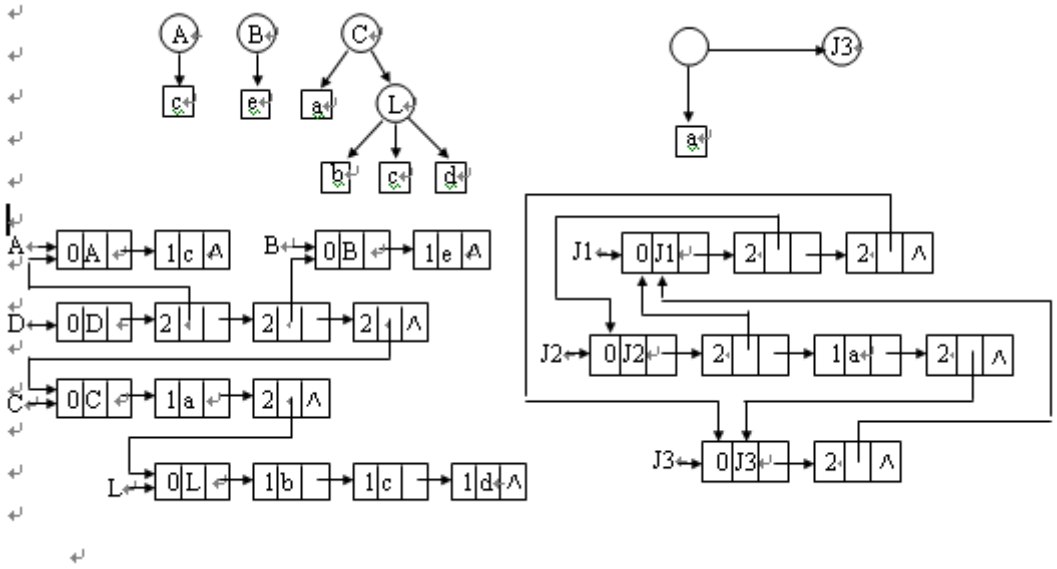
(6) Head (Head (Tail (Head (Tail (L6) ) ) ) ) )

5-7 画出下列广义表的图形表示和它们的存储表示：

(1) D(A(c), B(e), C(a, L(b, c, d)))

(2) J1(J2(J1, a, J3(J1)), J3(J1))

【解析】 (1) D(A(c), B(e), C(a, L(b, c, d))) (2) J1(J2(J1, a, J3(J1)), J3(J1))




三、应用题

5-1 实现复制广义表的递归算法。

【 解析 】

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;

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
```

5-2 如果矩阵 A 中存在这样的一个元素 A[i][j] 满足条件： A[i][j] 是第 i 行中值最小的元素， 且又是第 j 列中值最大的元素，则称之为该矩阵的一个马鞍点。编写一个程序计算出 m*n 的矩 阵 A 的所有马鞍点。

【解析】

```
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 ) 最小值是否鞍点

{ for(flag=1,k=0;k<m;k++)

if(min<A[k][j]) flag=0; 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
```

5-3 求广义表深度的递归算法

【解析】

```
int GList_Getdeph(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_Getdeph(L->ptr.hp)+1;

n=GList_Getdeph(L->ptr.tp);

return m>n?m:n;

}//GList_Getdeph
```

5-4 设 $A$ 和 $B$ 均为下三角矩阵，每一个都有 $n$ 行。因此在下三角区域中各有 $n(n+1)/2$ 个元素。另设有一个二维数组 $C$，它有 $n$ 行 $n+1$ 列。试设计一个方案，将两个矩阵 $A$ 和 $B$ 中的下三角区域元素存放于同一个 $C$ 中。要求将 $A$ 的下三角区域中的元素存放于 $C$ 的下三角区域中，$B$ 的下三角区域中的元素转置后存放于 $C$ 的上三角区域中。并给出计算 $A$ 的矩阵元素 $a\,ij$ 和 $B$ 的矩阵元素 $b\,ij$ 在 $C$ 中的存放位置下标的公式。

5-5 假设稀疏矩阵 A 采用三元组表示，编写一个函数计算其转置矩阵 B ，要求 B 也采用三元组表示。

【解析】

三元组表示中要求按行的顺序存放，所有转置过程不能直接将行下标和列下标转换，还必须使得列按顺序存放。因此在 A 中首先找出第一列中的所有元素，它们是转置矩阵第一行的非 0 元素，并把它们依次放在转置矩阵三元组数组 B 中；然后依次找出第二列中的所有元素，把它们依次放在数组 B 中；按照同样的方法逐列进行，直到找出第 n 列的所有元素，并把它们依次放在数组 B 中。实现本题功能的函数如下：

```
void transpose （ A ， B ）

smatrik A ， B ； /*A 是稀疏矩阵的三元组形式， B 是存放 A 的转置矩阵的三元组数组 */

int m ， n ， p ， q ， t ， col ；

/*m 为 A 中的行数； n 为 A 中的列数； t 为 A 中非 0 元素个数 */

/*q 为 B 的下一项位置； p 为 A 的当前项 */

m=A[0][0] ； n=A[0][1] ； t=A[0][2] ；

B[0][0]=n ； B[0][1]=m ； B[0][2]=t ； /* 产生第 0 行的结果 */

If （ t>0 ） /* 非 0 矩阵才做转置 */

{ q=1 ； for （ col=0 ； col<n ； col++ ） /* 按列转置 */

for （ p=1 ； p<=t ； p++ ） if （ A[p][1]==col ） { B[q][0]=A[p][1] ；

B[q][1]=A[p][0] ； B[q][2]=A[p][2] ； q++ ； } } }
```

一、选择题

1.若不考虑结点的数据信息的组合情况，具有 3 个结点的树共有种（ A ）形态，而二叉树共有( D )种形态。

A.2　　B.3

C.4　　D.5

2.对任何一棵二叉树，若 n0，n1，n2 分别是度为 0，1，2 的结点的个数，则 n0= ( C )

A.n1+1　　B.n1+n2

C.n2+1　　D.2n1+1

3.已知某非空二叉树采用顺序存储结构，树中结点的数据信息依次存放在一个一维数组中，即 ABC□DFE□□G□□H□□，该二叉树的中序遍历序列为 ( D )

A.G,D,B,A,F,H,C,E　　B.G,B,D,A,F,H,C,E

C.B,D,G,A,F,H,C,E　　D.B,G,D,A,F,H,C,E

4、具有 65 个结点的完全二叉树的高度为（ B ）。（根的层次号为 1）

A．8　　B．7　　C．6　　D．5

5、在有 N 个叶子结点的哈夫曼树中，其结点总数为（ D ）。

A 不确定　　B 2N　　C 2N+1　　D 2N-1

6、以二叉链表作为二叉树存储结构，在有 N 个结点的二叉链表中，值为非空的链域的个数为（ A ）。

A N-1　　B 2N-1　　C N+1　　D 2N+1

三、填空题。

1、对于一个具有 N 个结点的二叉树，当它为一颗 完全 二叉树时，具有最小高度。

2、对于一颗具有 N 个结点的二叉树，当进行链接存储时，其二叉链表中的指针域的总数为 2N 个， 其中 N-1 个用于链接孩子结点，N+1 个空闲着。

3、一颗深度为 K 的满二叉树的结点总数为 $2^k-1$ ，一颗深度为 K 的完全二叉树的结点总数的最小值为 $2^{k-1}$ ，最大值为 $2^k-1$ 。

4、已知一棵二叉树的前序序列为 ABDFCE，中序序列为 DFBACE，后序序列为 FDBECA 。

四、应用题。

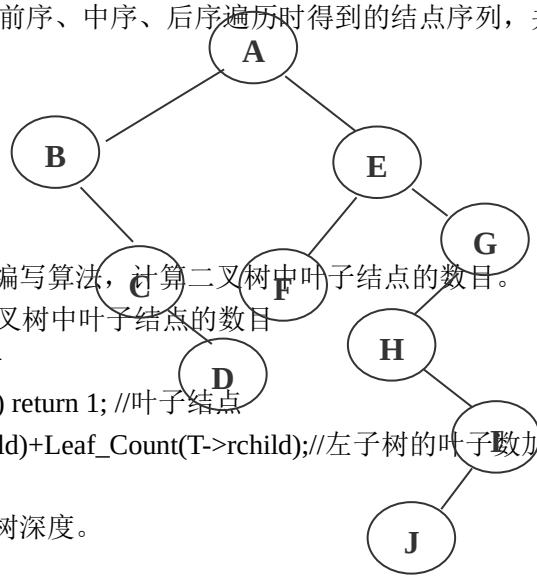
1、9．假设用于通讯的电文仅由 8 个字母组成，字母在电文中出现的频率分别为：

0.07，0.19，0.02，0.06，0.32，0.03，0.21，0.10

请为这 8 个字母设计哈夫曼编码。

解：仿照课本 148 页例 6-2。

2、已知一棵树二叉如下，请分别写出按前序、中序、后序遍历时得到的结点序列，并将该二叉树还原成森林。




解：参照课本 137~139 页。

先序遍历：ABCDEFGHIJ

中序遍历：BCDAFEHJIG

后序遍历：DCBFJIHG

五、算法设计题

1. 已知二叉树按照二叉链表方式存储，编写算法，计算二叉树中叶子结点的数目。

```
解：  int Leaf_Count (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
      }//Leaf_Count
```

2. 求二叉树中以值为 x 的结点为根的子树深度。

```
解：  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
```

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

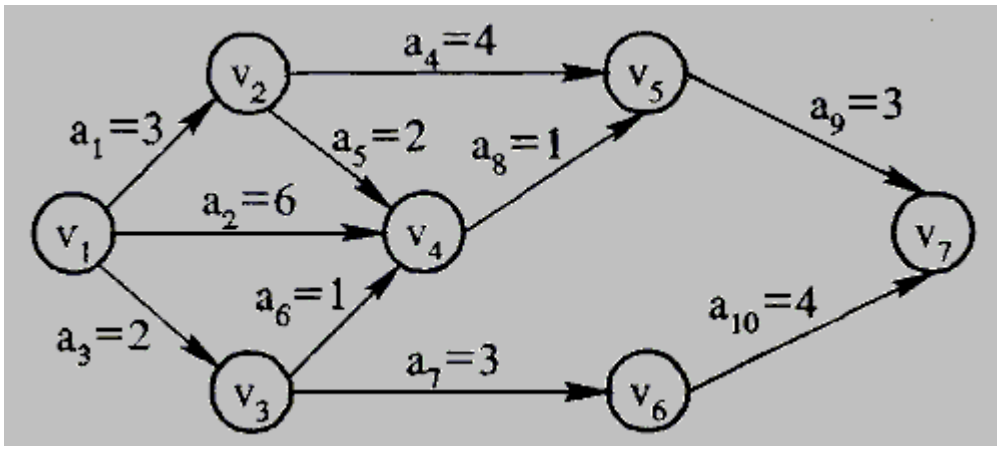
**第七章习题**

一、选择题

7-1 关键路径是指 AOE(Activity On Edge) 网中__C__。

A. 最长的回路 B. 最短的回路 C. 从源点到汇点 ( 结束顶点 ) 的最长路径 D. 从源点到汇点 ( 结束顶点 ) 的最短路径

7-2 已知 AOE 网中顶点 v1 ～ v7 分别表示 7 个事件，弧 al ～ a10 分别表示 10 个活动，弧上的数值表示每个活动花费的时间，如下图所示。那么，该网的关键路径的长度为 __（ 1 ）C__，活动 a6 的活动的（最迟开始时间 - 活动的最早开始时间）为 __（ 2 ）A__。




(1) A. 7 B. 9 C. 10 D. 11
(2) A. 3 B. 2 C. 1 D. 0

7-3 任何一个无向连通图的最小生成树 __A__。

A、只有一棵 B、有一棵或多棵

C、一定有多棵 D、可能不存在

7- 4 下面关于图的存储的叙述中正确的是 __A__。

A）邻接矩阵占用的存储空间只与图中结点个数有关，而与边数无关；

B）邻接矩阵占用的存储空间只与图中边数有关，而与结点个数无关；

C）邻接表占用的存储空间只与图中结点个数有关，而与边数无关；

D）邻接表占用的存储空间只与图中边数有关，而与结点个数无关。

7-5、在一个无向图中，所有顶点的度数之和等于所有边数的__B___倍。

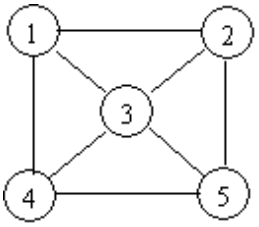
A．3　　B．2　　C．1　　D．1/2

7.6. 在一个具有 6 个顶点的有向图中，所有顶点的入度数之和与所有顶点的出度数之和的差等于__B__。

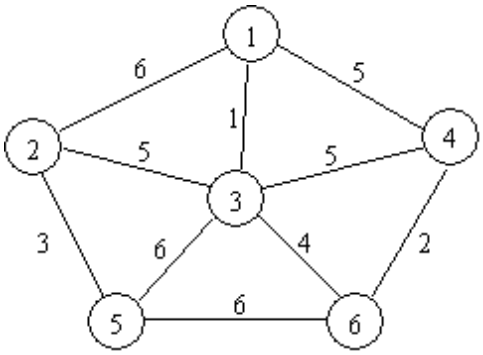
A．1　　B．0　　C．12　　D．6

二、简答题

7-1 给出如下图所示的无向图 G 的邻接矩阵和邻接表两种存储结构。并写出按此邻接表由顶点 1 出发进行深度优先遍历时得到的顶点序列，进行广度优先遍历时得到的顶点序列。




7-2 使用普里姆算法和克鲁斯卡尔算法构造出如图 3 所示的图 G 的最小生成树。




解：普里姆算法：课本 173~174 页

克鲁姆算法：课本 175~176 页

7-3 有 n 个顶点的无向连通图至少有多少条边？有 n 个顶点的有向强连通图至少有多少条边？试举例说明。

解：有 n 个顶点的无向连通图至少有 n-1 条边；

有 n 个顶点的有向强连通图至少有 n 条边。

三 算法题

*7.5　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。

第九章　查找表

1．设有 100 个数据元素，采用折半搜索时，最大比较次数为__7__。

3、在含有 n 个元素的顺序表中，用顺序查找方法，查找成功的平均查找长度为 __( n+1)/2__。

4、对有 14 个数据元素的有序表 R[0…13]进行折半搜索，搜索到 R[3]的关键码等于给定值，此时元素比较顺序依次为（ C ）。

A．R[0]，R[1]，R[2]，R[3]　　B．R[0]，R[13]，R[2]，R[3]

C．R[6]，R[2]，R[4]，R[3]　　D．R[6]，R[4]，R[2]，R[3]

通常查找线性表数据元素的方法有 二分法查找 和 顺序查找 两种方法，其中 二分法查找 是一种只适合于顺序存储结构 但效率更高的线性查找 的方法；而 顺序查找 是一种对顺序和链式存储结构均适用的方法。

6 散列法存储的基本思想是根据 关键码值 来决定 存储地址 ，冲突指的是 不同 关键字对应到相同的存储地址，处理冲突的主要方法有 开放定址法、链地址法、在哈希法、建立一个公共溢出区。

7 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。三次比较成功的结点有哪些？。

8 选取哈希函数 *H(k)= k*%11，用线性探测再散列法(及链地址法)处理冲突。试在 0～10 的散列地址空间中，对关键字序列（22，41，53，46，30，13，01，67）构造哈希表，并求等概率情况下查找成功的平均查找长度。

*9．写出折半查找的递归算法。

```
int Search_Bin1(SSTable ST,KeyType key,int low,int high)//递归
 { // 在有序表 ST 中折半查找其关键字等于 key 的数据元素。若找到，则函数值为
   // 该元素在表中的位置，否则为 0。算法 9.2
   int mid;
   // 置区间初值
   if (low<=high)
   {mid=(low+high)/2;
     if EQ(key,ST.elem[mid].key)  // 找到待查元素
       return mid;
     else if LT(key,ST.elem[mid].key)
       return   Search_Bin1( ST, key, low,mid-1); // 继续在前半区间进行查找
     else
       return   Search_Bin1( ST, key, mid+1,high); // 继续在后半区间进行查找
   }
   return 0; // 顺序表中不存在待查元素
 }
```

第十章 排序

1. 一个记录的关键字为（46，79，56，38，42，88），采用快速排序以第一个记录为基准得到的第一次划分结果为 [ C ]

A. (38, 42, 46, 56, 79, 88)　　B. (42, 38, 46, 79, 56, 88)

C. (42, 38, 46, 56, 79, 88)　　D. ( 38, 42, 46, 79, 56 , 88)

2. 下列排序方法中, 排序趟数与序列的原始状态有关的方法是 [ D ]

A. 选择排序　　B. 希尔排序　　C. 堆排序　　D. 冒泡排序

3、对 5 个不同的数据元素进行直接插入排序，最多需要进行 14 次比较。

4、简单选择排序算法在最好情况下所作的交换元素的次数为 0 。

5、具有 n 个记录的序列，采用冒泡排序，比较的次数最少是 n-1 次。

6. 堆排序实际上是一种 选择 排序，它的一个基本问题是如何建堆；对含 n 个元素的序列进行排序时，其时间复杂度是 O（nlogn）。

7. 快速排序在最坏情况下的时间复杂度是 $O(n^2)$。

8.若对序列(49,38,27,13,97,76,50,65)采用泡排序法(按照值的大小从小到大)进行排序，请分别在下表中写出每一趟排序的结果。

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 原 始 序 列 | 49 | 38 | 27 | 13 | 97 | 76 | 50 | 65 |
| 第 1 趟的结果 | 38 | 37 | 13 | 49 | 76 | 50 | 65 | 97 |
| 第 2 趟的结果 | 37 | 13 | 38 | 49 | 50 | 65 | 76 | |
| 第 3 趟的结果 | 13 | 37 | 38 | 49 | 50 | 65 | | |

9、有一组关键字{50，52，85，22，96，17，36，55}，请用快速排序，写出第一趟结果。

解：｛ 36，17，22，50，96，85，52，55 ｝

10、设待排序序列为 {10, 18, 4, 3, 6, 12, 1, 9, 15, 8}，请给出用希尔排序每一趟的结果。增量序列取为 5, 3, 2, 1。

解：第一躺：{ 10，1，4，3，6，12，18，9，15，8 }

第二趟：{ 3，1，4，8，6，12，10，9，15，18 }

第三趟：{ 3，1，4，8，6，9，10，12，15，18 }

第四躺：{ 1，3，4，6，8，9，10，12，15，18 }

第二章

```
1 Status Insert_SqList(SqList &va,int x)//把 x 插入递增有序表 va 中
{
 if(va.length+1>va.listsize) return ERROR;
 va.length++;
 for(i=va.length-1;va.elem[i]>x&&i>=0;i--)
   va.elem[i+1]=va.elem[i];
 va.elem[i+1]=x;
 return OK;
}//Insert_SqList
```

```
2. 1 void reverse(SqList &A)//顺序表的就地逆置
{
 for(i=0,j=A.length-1;i<j;i++,j--)
   A.elem[i]<->A.elem[j];
}//reverse
```

```
2.2 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
```

分析:本算法的思想是,逐个地把 L 的当前元素 q 插入新的链表头部,p 为新表表头.

```
3 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
```

第三章

```
5 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  ERROR;//队列已空
p=Q->next->next;
x=p->data;
Q->next->next=p->next;
free(p);
return OK;
}//DeCiQueue
```

第四章

```
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++;
  //在结构数组 T 中逐元素查找当前字符 c 是否已记录过.
  //当循环停止时，再看是什么原因造成的停止。
  if(T[j].ch) T[j].num++;
  //循环停止时 T[j].ch 不等于 NULL,说明是由于 T[j].ch=c 所致
  //若是 T[j].ch =c 所致则说明字符 c 在串 s 中已经出现过
  //故将其个数加 1.
  else T[j]={c,1}; //否则为首次出现，将其出现次数记为 1.
  }//for
  for(j=0;T[j].ch;j++) //打印每个字符在串 s 中的出现次数。
  printf(“%c: %d\n”,T[j].num);
  }//StrAnalyze
```

第五章

```
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 ) 最小值是否鞍点

{ for(flag=1,k=0;k<m;k++)

if(min<A[k][j]) flag=0;

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
```

第六章

五、1 int Leaf_Count (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
}//Leaf_Count
```

2

```
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
```