第一章

三、计算下列程序段中 X=X+1 的语句频度

```
for(i=1;i<=n;i++)
   for(j=1;j<=i;j++)
for(k=1;k<=j;k++)
 x=x+1;
```

[提示]：

i=1 时： $1 = (1+1)\times1/2 = (1+1^2)/2$

i=2 时： $1+2 = (1+2)\times2/2 = (2+2^2)/2$

i=3 时： $1+2+3 = (1+3)\times3/2 = (3+3^2)/2$

…

i=n时： $1+2+3+\ldots\ldots+n = (1+n)\times n/2 = (n+n^2)/2$

$f(n) = [\ (1+2+3+\ldots\ldots+n) + (1^2 + 2^2 + 3^2 + \ldots\ldots + n^2)\ ] / 2$

$=[\ (1+n)\times n/2 + n(n+1)(2n+1)/6\ ] / 2$

$=n(n+1)(n+2)/6$

$=n^3/6+n^2/2+n/3$

区分语句频度和算法复杂度：

$O(f(n)) = O(n^3)$

四、试编写算法求一元多项式$Pn(x)=a_0+a_1x+a_2x^2+a_3x^3+\ldots a_nx^n$的值$P_n(x_0)$，并确定算法中的每一语句的执行次数和整个算法的时间复杂度，要求时间复杂度尽可能的小，规定算法中不能使用求幂函数。注意：本题中的输入a_i(i=0,1,…,n), x和n，输出为$P_n(x_0)$.通常算法的输入和输出可采用下列两种方式之一：

（1）通过参数表中的参数显式传递；

（2）通过全局变量隐式传递。

试讨论这两种方法的优缺点，并在本题算法中以你认为较好的一种方式实现输入和输出。

[提示]：float PolyValue(float a[ ], float x, int n) {……}

核心语句：

```
p=1;  (x 的零次幂)
s=0;
i 从 0 到 n 循环
s=s+a[i]*p;
p=p*x;
```

或：

```
p=x;  (x 的一次幂)
s=a[0];
i 从 1 到 n 循环
s=s+a[i]*p;
p=p*x;
```

第二章

2.1 描述以下三个概念的区别：头指针，头结点，首元素结点。

2.2 填空：

（1） 在顺序表中插入或删除一个元素，需要平均移动____一半____元素，具体移动的元素个数与____插入或删除的位置____有关。

（2） 在顺序表中，逻辑上相邻的元素，其物理位置_____相邻。在单链表中，逻辑上相邻的元素，其物理位置_____相邻。

（3） 在带头结点的非空单链表中，头结点的存储位置由_____指示，首元素结点的存储位置由_____指示，除首元素结点外，其它任一元素结点的存储位置由____其直接前趋的 next 域____指示。

2.3 已知 L 是无表头结点的单链表，且 P 结点既不是首元素结点，也不是尾元素结点。按要求从下列语句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：（4）、（1）。

b. 在P结点前插入S结点的语句序列是：（7）、（11）、（8）、（4）、（1）。

c. 在表首插入S结点的语句序列是：（5）、（12）。

d. 在表尾插入S结点的语句序列是：（11）、（9）、（1）、（6）。

供选择的语句有：

（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;

2.4 已知线性表 L 递增有序。试写一算法，将 X 插入到 L 的适当位置上，以保持线性表 L 的有序性。

[提示]：void insert(SeqList *L; ElemType x)

< 方法 1 >

（1）找出应插入位置 i，（2）移位，（3）……

< 方法 2 > 参 P. 229

2.5 写一算法，从顺序表中删除自第 i 个元素开始的 k 个元素。

[提示]：注意检查 i 和 k 的合法性。

（集体搬迁，“新房”、“旧房”）

< 方法 1 > 以待移动元素下标 m（“旧房号”）为中心，
计算应移入位置（“新房号”）：

```
for ( m= i－1+k;  m<= L－>last;  m++)
    L－>elem[ m－k ] = L－>elem[ m ];
```

< 方法 2 > 同时以待移动元素下标 m 和应移入位置 j 为中心：

< 方法 2 > 以应移入位置 j 为中心，计算待移动元素下标：

2.6 已知线性表中的元素（整数）以值递增有序排列，并以单链表作存储结构。试写一高效算法，删除表中所有大于 mink 且小于 maxk 的元素（若表中存在这样的元素），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，它们的值为任意的整数）。

[提示]：注意检查 mink 和 maxk 的合法性：mink < maxk

不要一个一个的删除（多次修改 next 域）。

（1） 找到第一个应删结点的前驱 pre

```
pre=L;   p=L－>next;
while (p!=NULL && p－>data <= mink)
   { pre=p;  p=p－>next;  }
```

（2） 找到最后一个应删结点的后继 s，边找边释放应删结点

```
s=p;
while (s!=NULL && s－>data < mink)
```

{ t =s; s=s－>next; free(t); }
（3） pre－>next = s;

2.7 试分别以不同的存储结构实现线性表的就地逆置算法，即在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）。

（1） 以一维数组作存储结构，设线性表存于 a(1:arrsize)的前 elenum 个分量中。

（2） 以单链表作存储结构。

[方法 1]:在原头结点后重新头插一遍

[方法 2]:可设三个同步移动的指针 p, q, r，将 q 的后继 r 改为 p

2.8 假设两个按元素值递增有序排列的线性表 A 和 B，均以单链表作为存储结构，请编写算法，将 A 表和 B 表归并成一个按元素值递减有序的排列的线性表 C，并要求利用原表（即 A 表和 B 表的）结点空间存放表 C.

[提示]：参 P.28 例 2-1

< 方法 1 >

```
void  merge(LinkList  A;  LinkList  B;  LinkList  *C)
{  ……
pa=A－>next;  pb=B－>next;
*C=A;  (*C)－>next=NULL;
while ( pa!=NULL && pb!=NULL )
  { if ( pa－>data <= pb－>data )
{ smaller=pa;  pa=pa－>next;
 smaller－>next = (*C)－>next;     /*  头插法  */
 (*C)－>next = smaller;
}
else
{ smaller=pb;  pb=pb－>next;
 smaller－>next = (*C)－>next;
 (*C)－>next = smaller;
}

while ( pa!=NULL)
{  smaller=pa;  pa=pa－>next;
  smaller－>next = (*C)－>next;
  (*C)－>next = smaller;
}
while ( pb!=NULL)
{  smaller=pb;  pb=pb－>next;
  smaller－>next = (*C)－>next;
  (*C)－>next = smaller;
}
```

< 方法 2 >

```
LinkList  merge(LinkList  A;  LinkList  B)
{  ……
LinkList  C;
pa=A－>next;  pb=B－>next;
C=A;  C－>next=NULL;
……
……
  return  C;
```

2.9 假设有一个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 所指结点的前趋结点。

[提示]：设指针 p 指向 s 结点的前趋的前趋，则 p 与 s 有何关系？

2.10 已知有单链表表示的线性表中含有三类字符的数据元素（如字母字符、数字字符和其它字符），试编写算法来构造三个以循环链表表示的线性表，使每个表中只含同一类的字符，且利用原表中的结点空间作为这三个表的结点空间，头结点可另辟空间。

2.11 设线性表A=(a_1, a_2,…,a_m)，B=(b_1, b_2,…,b_n)，试写一个按下列规则合并A、B为线性表C的算法，使得：

C= (a_1, b_1,…,a_m, b_m, b_{m+1}, …,b_n)　　当m≤n时；

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

线性表 A、B、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。

[提示]：void　merge(LinkList　A;　LinkList　B;　LinkList　*C)

　　或：LinkList　merge(LinkList　A;　LinkList　B)

2.12 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来构成这两个链表。

[提示]：注明用头指针还是尾指针。

2.13 建立一个带头结点的线性链表，用以存放输入的二进制数，链表中每个结点的 data 域存放一个二进制位。并在此链表上实现对二进制数加 1 的运算 。

[提示]：可将低位放在前面。

2.14 设多项式 P(x)采用课本中所述链接方法存储。写一算法，对给定的 x 值，求 P(x)的值。

[提示]：float　PolyValue(Polylist　p;　float　x) {……}

第三章

1. 按图 3.1(b)所示铁道（两侧铁道均为单向行驶道）进行车厢调度，回答：

⑴ 如进站的车厢序列为 123，则可能得到的出站车厢序列是什么？　123、213、132、231、321（312）

⑵如进站的车厢序列为 123456，能否得到 435612 和 135426 的出站序列，并说明原因。（即写出以“S”表示进栈、以“X”表示出栈的栈操作序列）。

SXSS XSSX XXSX　或　S1X1S2S3X3S4S5X5X4X2S6X6

2. 设队列中有 A、B、C、D、E 这 5 个元素，其中队首元素为 A。如果对这个队列重复执行下列 4 步操作：

（1）　输出队首元素；

（2）　把队首元素值插入到队尾；

（3）　删除队首元素；

（4）　再次删除队首元素。

直到队列成为空队列为止，则是否可能得到输出序列：

（1）　A、C、E、C、C　　（2）　A、C、E

（3）　A、C、E、C、C、C　　（4）　A、C、E、C

[提示]：

A、B、C、D、E　（输出队首元素 A）

A、B、C、D、E、A　（把队首元素 A 插入到队尾）

B、C、D、E、A　（删除队首元素 A）

C、D、E、A　（再次删除队首元素 B）

C、D、E、A （输出队首元素 C）

C、D、E、A、C　（把队首元素 C 插入到队尾）

D、E、A、C　（删除队首元素 C）

E、A、C　（再次删除队首元素 D）

3. 给出栈的两种存储结构形式名称，在这两种栈的存储结构中如何判别栈空与栈满？

4. 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

5. 试写一个算法，判断依次读入的一个以@为结束符的字母序列，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。

[提示]：

（1） 边读边入栈，直到&

（2） 边读边出栈边比较，直到……

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

[提示]：

例：

中缀表达式：a+b 后缀表达式: ab+

中缀表达式：a+b×c 后缀表达式: abc×+

中缀表达式：a+b×c-d 后缀表达式: abc×+d-

中缀表达式：a+b×c-d/e 后缀表达式: abc×+de/-

中缀表达式：a+b×(c-d)-e/f 后缀表达式: abcd-×+ef/-

- 后缀表达式的计算过程：（简便）

顺序扫描表达式，

（1） 如果是操作数，直接入栈；

（2） 如果是操作符 op，则连续退栈两次，得操作数 X, Y，计算 X op Y，并将结果入栈。

- 如何将中缀表达式转换为后缀表达式？

顺序扫描中缀表达式，

（1）如果是操作数，直接输出；

（2）如果是操作符op_2，则与栈顶操作符op_1比较：

如果$op_2 > op_1$，则op_2入栈；

如果$op_2 = op_1$，则脱括号；

如果$op_2 < op_1$，则输出op_1；

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、入队列和出队列的算法。

[提示]： 参 P.56 P.70 先画图.

```
typedef  LinkList  CLQueue;
int InitQueue(CLQueue * Q)
int EnterQueue(CLQueue Q, QueueElementType x)
int DeleteQueue(CLQueue Q, QueueElementType *x)
```

8. 要求循环队列不损失一个空间全部都能得到利用，设置一个标志域 tag，以 tag 为 0 或 1 来区分头尾指针相同时的队列状态的空与满，请编写与此结构相应的入队与出队算法。

[提示]：

初始状态：front==0, rear==0, tag==0

队空条件：front==rear, tag==0

队满条件：front==rear, tag==1

其它状态：front !=rear, tag==0（或 1、2）

入队操作：

…

…（入队）

if (front==rear) tag=1；（或直接 tag=1）

出队操作：
…
…（出队）
tag=0；

[问题]：如何明确区分队空、队满、非空非满三种情况？

9. 简述以下算法的功能（其中栈和队列的元素类型均为 int）：
（1）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。
（2）void proc_2(Stack S, int e)
{ Stack T; int d;
InitStack(&T);
while(!EmptyStack(S))
{ Pop(&S, &d);
if (d!=e) Push(&T, d);
}
while(!EmptyStack(T))
{ Pop(&T, &d);
Push(&S, d);
}
}
删除栈 S 中所有等于 e 的元素。
（3）void proc_3(Queue *Q)
{ Stack S; int d;
InitStack(&S);
while(!EmptyQueue(*Q))
{
DeleteQueue(Q, &d);
Push(&S, d);
}
while(!EmptyStack(S))
{ Pop(&S, &d);
EnterQueue(Q，d)
}
}
将队列 Q 逆序。

第四章

1. 设 s='I AM A STUDENT', t='GOOD', q='WORKER'。给出下列操作的结果：
StrLength(s); SubString(sub1,s,1,7); SubString(sub2,s,7,1);
StrIndex(s,'A',4); StrReplace(s,'STUDENT',q);
StrCat(StrCat(sub1,t), StrCat(sub2,q));
[参考答案]
StrLength(s)=14; sub1= 'I AM A_'; sub2= '_'; StrIndex(s,'A',4)=6;
StrReplace(s,'STUDENT',q)= 'I AM A WORKER';
StrCat(StrCat(sub1,t), StrCat(sub2,q))= 'I AM A GOOD WORKER';

2. 编写算法，实现串的基本操作 StrReplace(S,T,V)。
3. 假设以块链结构表示串，块的大小为 1，且附设头结点。
试编写算法，实现串的下列基本操作：
StrAsign(S,chars)； StrCopy(S,T)； StrCompare(S,T)； StrLength(S)； StrCat(S,T)；
SubString(Sub,S,pos,len)。
[说明]：用单链表实现。

4. 叙述以下每对术语的区别：空串和空格串；串变量和串常量；主串和子串；串变量的名字和串变量的值。
5. 已知：S=”(xyz)*”,T=”(x+z)*y”。试利用联接、求子串和置换等操作，将 S 转换为 T.
6. S 和 T 是用结点大小为 1 的单链表存储的两个串，设计一个算法将串 S 中首次与 T 匹配的子串逆置。
7. S 是用结点大小为 4 的单链表存储的串,分别编写算法在第 k 个字符后插入串 T，及从第 k 个字符删除 len 个字符。

以下算法用定长顺序串：

8. 写下列算法：

（1）将顺序串 r 中所有值为 ch1 的字符换成 ch2 的字符。
（2）将顺序串 r 中所有字符按照相反的次序仍存放在 r 中。
（3）从顺序串 r 中删除其值等于 ch 的所有字符。
（4）从顺序串 r1 中第 index 个字符起求出首次与串 r2 相同的子串的起始位置。
（5）从顺序串 r 中删除所有与串 r1 相同的子串。

9. 写一个函数将顺序串 s1 中的第 i 个字符到第 j 个字符之间的字符用 s2 串替换。

[提示]：（1）用静态顺序串 （2）先移位，后复制

10. 写算法，实现顺序串的基本操作 StrCompare(s,t)。
11. 写算法，实现顺序串的基本操作 StrReplace(&s,t,v)。

[提示]：

（1） 被替换子串定位（相当于第 9 题中 i）
（2） 被替换子串后面的字符左移或右移（为替换子串准备房间）
（3） 替换子串入住（复制）
（4） 重复上述，直到……

第五章

1. 假设有 6 行 8 列的二维数组 A，每个元素占用 6 个字节，存储器按字节编址。已知 A 的基地址为 1000，计算：

（1） 数组 A 共占用多少字节； （288）
（2） 数组 A 的最后一个元素的地址； （1282）
（3） 按行存储时，元素A_{36}的地址； （1126）
（4） 按列存储时，元素A_{36}的地址； （1192）

[注意]：本章自定义数组的下标从 1 开始。

2. 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(1:3n-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。

i = k/3 + 1， j = k%3 + i - 1 = k%3 + k/3
或：
i = k/3 + 1， j = k - 2×(k/3)

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

[提示]：参考 P.28 例、P.47 例。

4. 在稀疏矩阵的快速转置算法 5.2 中，将计算 position[col]的方法稍加改动，使算法

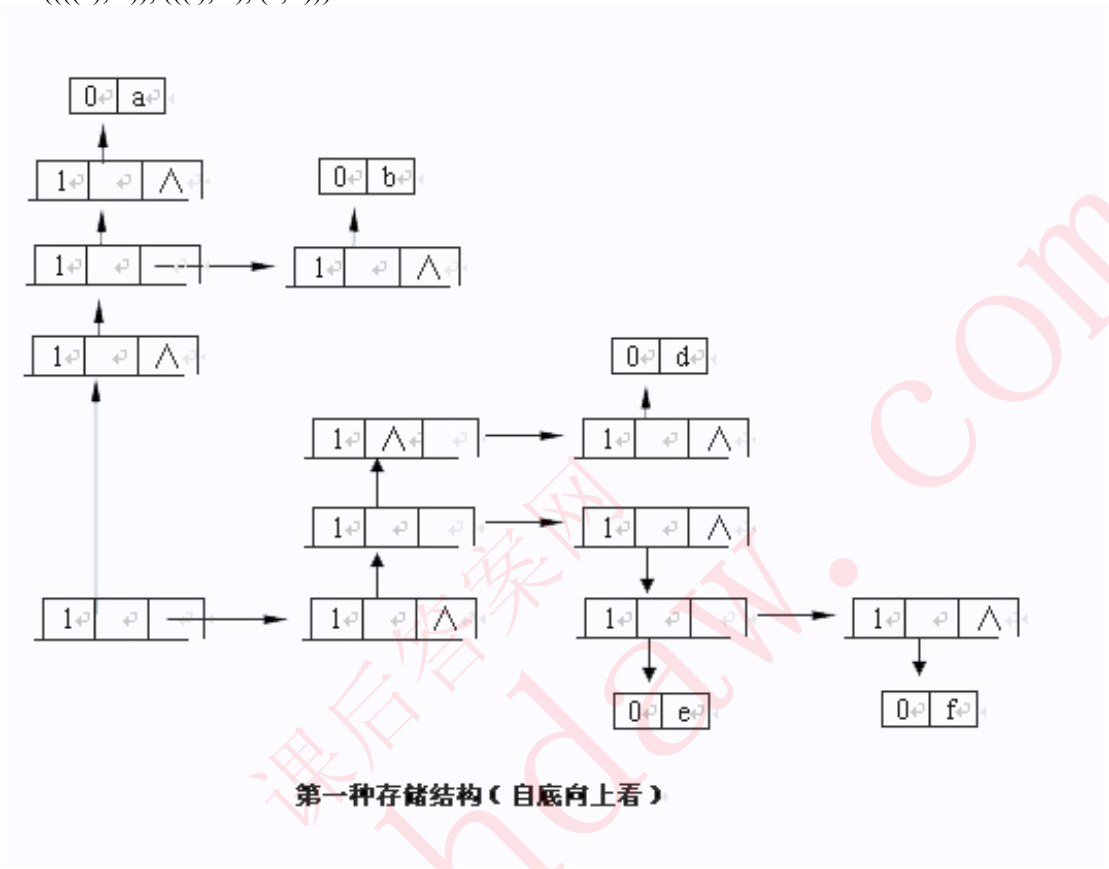
只占用一个辅助向量空间。

5．写一个在十字链表中删除非零元素a_{ij}的算法。

[提示]：“删除”两次，释放一次。

6．画出下面广义表的两种存储结构图示：

((((a), b)), ((( ), d), (e, f)))



第一种存储结构（自底向上看）

7．求下列广义表运算的结果：

（1） HEAD[((a,b),(c,d))];

（2） TAIL[((a,b),(c,d))];

（3） TAIL[HEAD[((a,b),(c,d))]];

（4） HEAD[TAIL[HEAD[((a,b),(c,d))]]]; b

（5） TAIL[HEAD[TAIL[((a,b),(c,d))]]]; (d)

第六章

1．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。

2．对题 1 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。

3．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，则该树中有多少个叶子结点？

[提示]：参考 P.116 性质 3

∵ $n=n_0+n_1+\cdots\cdots+n_k$

$B=n_1+2n_2+3n_3+\cdots\cdots+kn_k$

$n=B+1$

∴ $n_0+n_1+\cdots\cdots+n_k=n_1+2n_2+3n_3+\cdots\cdots+kn_k+1$

∴ $n_0=n_2+2n_3+\cdots\cdots+(k-1)n_k+1$

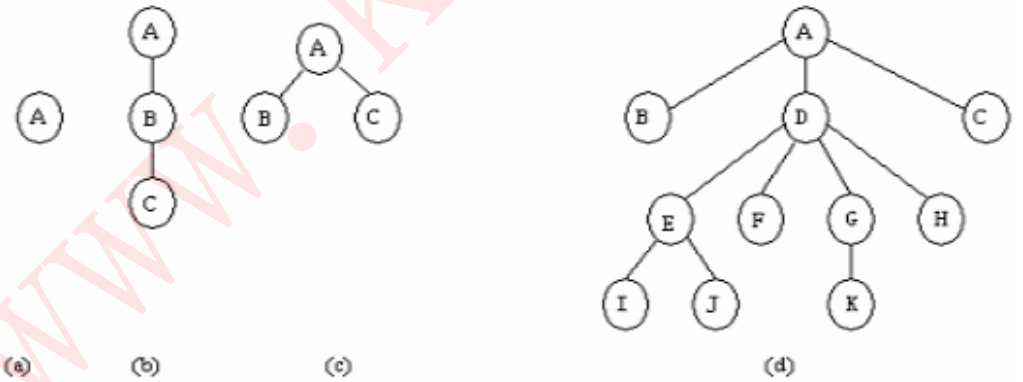
4.假设一棵二叉树的先序序列为 EBADCFHGIKJ，中序序列为 ABCDEFGHIJK，请画出该二叉树。

[提示]：参考 P.148

6. 已知二叉树有 50 个叶子结点，则该二叉树的总结点数至少应有多少个？
[提示]：一个叶子结点，总结点数至多有多少个？可压缩一度结点。

7. 给出满足下列条件的所有二叉树：
   a) 前序和中序相同
   b) 中序和后序相同
   c) 前序和后序相同
[提示]：去异存同。
   a) D L R 与 L D R 的相同点：D R，如果无 L，则完全相同，如果无 LR，…。
   b) L D R 与 L R D 的相同点：L D，如果无 R，则完全相同。
   c) D L R 与 L R D 的相同点：D，如果无 L R，则完全相同。
（如果去 D，则为空树）
7．n 个结点的 K 叉树，若用具有 k 个 child 域的等长链结点存储树的一个结点，则空的 Child 域有多少个？
[提示]：参考 P.119

8．画出与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。
[提示]：
（1）先画出对应的二叉树
（2）树的后根序列与对应二叉树的中序序列相同
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
（1）请为这 8 个字母设计哈夫曼编码，
（2）求平均编码长度。
10．已知二叉树采用二叉链表存放,要求返回二叉树 T 的后序序列中的第一个结点的指针,是否可不用递归且不用栈来完成?请简述原因.
[提示]：无右子的“左下端”
11．画出和下列树对应的二叉树：



12．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。
13．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。
[提示]：
[方法 1]：（1）按先序查找；（2）超前查看子结点（3）按后序释放；

```
void DelSubTree(BiTree   *bt,   DataType x)
{
if ( *bt != NULL && (*bt) ->data==x )
{   FreeTree(*bt);
```

```
*bt =NULL;
}
else    DelTree( *bt,    x)

void DelTree(BiTree bt,    DataType x)
{ if ( bt )
     { if (bt->LChild && bt->LChild->data==x)
{ FreeTree(bt->LChild);
   bt->LChild=NULL;
}

        if (bt->RChild && bt->RChild->data==x)
{ FreeTree(bt->RChild);
   bt->RChild=NULL;
}

DelTree(bt->LChild,    x);
DelTree(bt->RChild,    x);
}
}
```

[方法 2]：（1）先序查找；（2）直接查看当前根结点（3）用指针参数；
[方法 3]：（1）先序查找；（2）直接查看当前根结点
（3）通过函数值，返回删除后结果；
（参示例程序）

14．分别写函数完成：在先序线索二叉树 T 中，查找给定结点*p 在先序序列中的后继。在后序线索二叉树 T 中，查找给定结点*p 在后序序列中的前驱。
[提示]：
（1）先查看线索，无线索时用下面规律：
（2）结点*p 在先序序列中的后继为其左子或右子；
（3）结点*p 在后序序列中的前驱也是其左子或右子。

15．分别写出算法，实现在中序线索二叉树中查找给定结点*p 在中序序列中的前驱与后继。（参例题）

16．编写算法，对一棵以孩子-兄弟链表表示的树统计其叶子的个数。
[提示]：
（1）可将孩子-兄弟链表划分为根、首子树、兄弟树，递归处理。
（2）可利用返回值，或全局变量。

17．对以孩子-兄弟链表表示的树编写计算树的深度的算法。

18．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后序遍历非递归的算法。（参课本）

19．设二叉树按二叉链表存放，写算法判别一棵二叉树是否是一棵正则二叉树。正则二叉树是指：在二叉树中不存在子树个数为 1 的结点。
[提示]：可利用任何递归、非递归遍历算法。

20．计算二叉树最大宽度的算法。二叉树的最大宽度是指：二叉树所有层中结点个数的最大值。
[提示]：
[方法一]：
（1）利用队列，初值为根
（2）出队访问，并将左、右子入队，直到队空
（3）记录每一层中最后一个结点在队中的位置

（4）第 i 层最后一个结点的右子，必是第 i+1 层的最后一个结点
（5）第 1 层最后一个结点在队中的位置为 0

[方法二]：利用层号和全局数组，任意遍历、统计

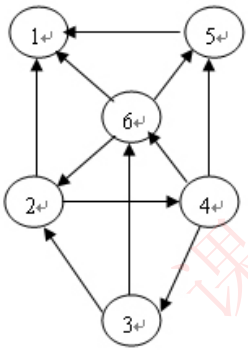
21. 已知二叉树按照二叉链表方式存储，利用栈的基本操作写出先序遍历非递归形式的算法。

22. 证明：给定一棵二叉树的前序序列与中序序列，可唯一确定这棵二叉树；
给定一棵二叉树的后序序列与中序序列，可唯一确定这棵二叉树；
23. 二叉树按照二叉链表方式存储，编写算法将二叉树左右子树进行交换。

第七章

7.1 已知如图所示的有向图，请给出该图的：

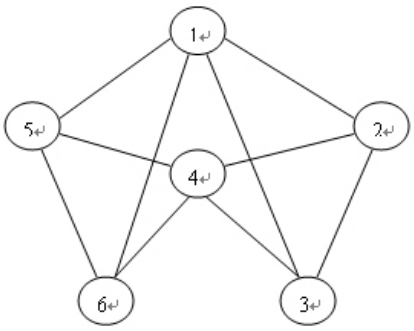
（1） 每个顶点的入度、出度；
（2） 邻接矩阵；
（3） 邻接表；
（4） 逆邻接表；
（5） 十字链表；
（6） 强连通分量。



题 1 图

7.2 已知如图所示的无向图，请给出该图的：

（1） 邻接多重表；（要求每个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。）
（2） 从顶点 1 开始，深度优先遍历该图所得顶点序列和边的序列；（给出深度优先搜索树）
（3） 从顶点 1 开始，广度优先遍历该图所得顶点序列和边的序列。（给出广度优先搜索树）



题 2 图

7.3 已知如图 7.31 所示的 AOE-网，试求：

（1） 每个事件的最早发生时间和最晚发生时间；
（2） 每个活动的最早开始时间和最晚开始时间；
（3） 给出关键路径。

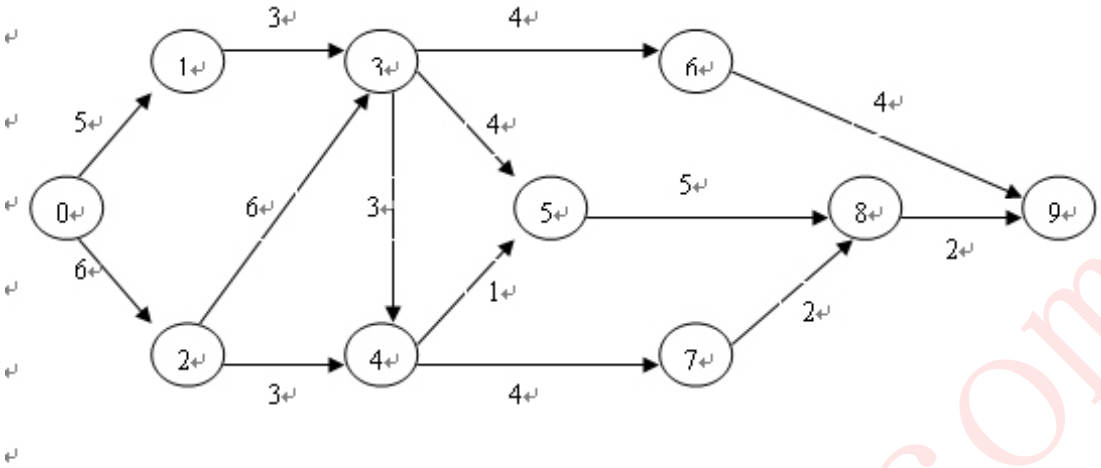


图 7.31 题 7.3 用图

7.4 已知如图 7.30 所示的有向网，试利用 Dijkstra 算法求顶点 1 到其余顶点的最短路径，并给出算法执行过程中各步的状态。

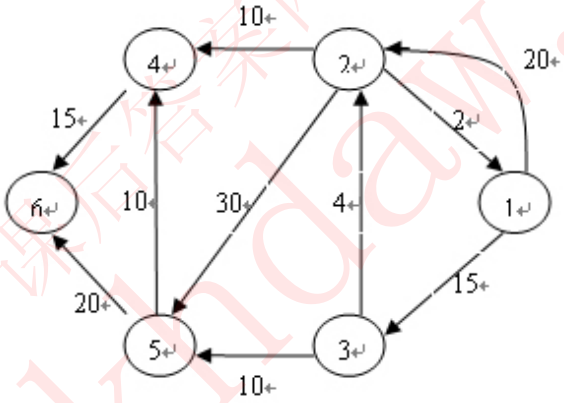


图 7.30 题 7.4 用图

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。

[参例题]

7.6 试在邻接矩阵存储结构上实现图的基本操作：InsertVertex(G,v); InsertArc(G, v, w); DeleteVertex(G,v)和 DeleteArc(G, v, w)。

7.7 试对邻接表存储结构重做题 6。

7.8 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。

7.9 同上题要求。试基于图的广度优先搜索策略写一算法。

7.10 试利用栈的基本操作，编写按深度优先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 看成是一种抽象数据类型。

7.11 采用邻接表存储结构，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（指顶点序列中不含有重现的顶点）的算法。

[提示]：利用深度搜索，增设一个深度参数，深度超过 k 则停止对该结点的搜索。

7.12 下图是带权的有向图 G 的邻接表表示法。从结点 V1 出发，深度遍历图 G 所得结点

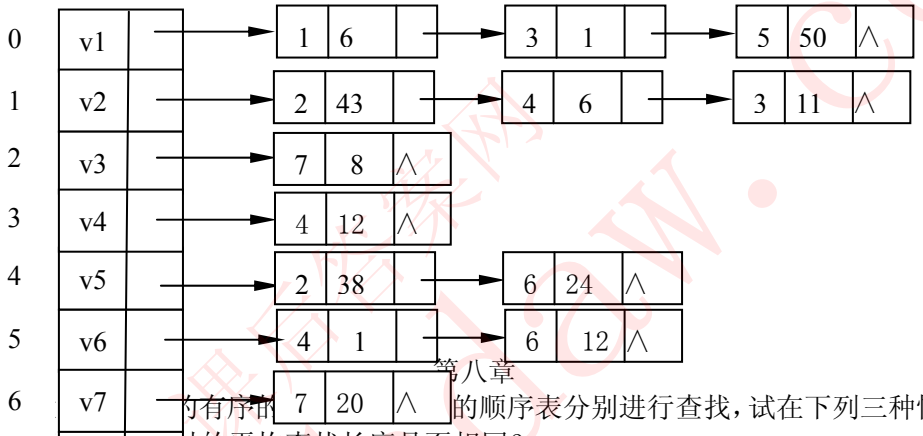
序列为（ A ），广度遍历图 G 所得结点序列为（ B ）；G 的一个拓扑序列是（ C ）；从结点 V1 到结点 V8 的最短路径为（ D ）；从结点 V1 到结点 V8 的关键路径为（ E ）。

其中 A、B、C 的选择有：

① V1,V2,V3,V4,V5,V6,V7,V8
② V1,V2,V4,V6,V5,V3,V7,V8
③ V1,V2,V4,V6,V3,V5,V7,V8
④ V1,V2,V4,V6,V7,V3,V5,V8
⑤ V1,V2,V3,V8,V4,V5,V6,V7
⑥ V1,V2,V3,V8,V4,V5,V7,V6
⑦ V1,V2,V3,V8,V5,V7,V4,V6

D、E 的选择有：

① V1,V2,V4,V5,V3,V8
② V1,V6,V5,V3,V8
③ V1,V6,V7,V8
④ V1,V2,V5,V7,V8



题 12 图

第八章

8.1 考[illegible]的有序[illegible]的顺序表分别进行查找，试在下列三种情况下分别讨论[illegible]时的平均查找长度是否相同？

（1） 查找成功，即表中没有关键字等于给定值 K 的记录。

（2） 查找成功，且表中只有一个关键字等于给定值 K 的记录。

（3） 查找成功，且表中有若干[illegible]等于给定值 K 的记录，一次查找要求找出所有记录。

[提示]：均用顺序查找法。

8.2 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。

[提示]：根据 P.191 ASL 定义计算平均查找长度。

8.3 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。

[提示]：沿最左分支生长，前提是：其任一祖先的右分支与左分支等深，如不等深，则先生长右分支，而生长右分支的前提是：其任一祖先的左分支与右分支等深，如不等深，则先生长左分支，如此交互考虑，逐步生长，直到 12 个结点。

8.4 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，画出每一步执行后 2-3 树的状态。

8.5 选取哈希函数 $H(k)=(3k)\%11$，用线性探测再散列法处理冲突。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与不成功时的平均查找长度。

[提示]：平均查找长度参考 P.225。

0	1	2	3	4	5	6	7	8	9	10

22		41	30	01	53	46	13	67		
1		1	2	2	1	1	2	6		

$ASL_{succ} = (1\times4 + 2\times3 + 6) / 8 = 2$
$ASL_{unsucc} = (2 + 1 + 8 + 7 + 6 + 5 + 4 + 3 + 2 + 1 + 1) / 11 = 40 / 11$

8.6 试为下列关键字建立一个装载因子不小于 0.75 的哈希表，并计算你所构造的哈希表的平均查找长度。
（ZHAO，QIAN，SUN，LI，ZHOU，WU，ZHENG，WANG，CHANG，CHAO，YANG，JIN）
[提示]：（1）由装填因子求出表长，（2）可利用字母序号设计哈希函数，
（3）确定解决冲突的方法。

8.7 试编写利用折半查找确定记录所在块的分块查找算法。

8.8 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。
[提示]：检验中序遍历序列是否递增有序？
[方法 1]：用非递归中序遍历算法，设双指针 pre， p
一旦 pre->data > p->data 则返回假
[方法 2]：用递归中序遍历算法，设全局指针 pre，（参中序线索化算法）
一旦 pre->data > p->data 则返回假
[方法 3]：用递归（中序或后序）算法
（1）空树是（2）单根树是（3）左递归真（4）右递归真
（5）左子树的右端小于根（6）右子树的左端大于根

8.9 编写算法，求出指定结点在给定的二叉排序树中所在的层数。

8.10 编写算法，在给定的二叉排序树上找出任意两个不同结点的最近公共祖先（若在两结点 A、B 中，A 是 B 的祖先，则认为 A 是 A、B 的最近公共祖先）。
[提示]：
（1） 假设 A<=B，
（2） 从根开始，左走或右走，直到 A 在左（或根），B 在右（或根）。

8.11 编写一个函数，利用二分查找算法在一个有序表中插入一个元素 x，并保持表的有序性。
[提示]：（1）表中已经有 x，（2）表中没有 x
参 P. 231

8.12 已知长度为 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，画出插入完成后的二叉排序树并求其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） 按表中元素的顺序依次构造一棵平衡二叉排序树，并求其等概率的情况下查找成功的平均查找长度。
[提示]：画出判定树或排序树，根据 P.191 ASL 定义计算平均查找长度。

8.13 含有 9 个叶子结点的 3 阶 B-树中至少有多少个非叶子结点？含有 10 个叶子结点的 3 阶 B-树中至少有多少个非叶子结点？

[提示]：叶子结点对应空指针。

8.14 写一时间复杂度为$O(\log_2 n + m)$的算法，删除二叉排序树中所有关键字不小于x的结点，并释放结点空间。其中n为树中的结点个数，m为被删除的结点个数。

[提示]：

（1） p=root

（2） 如果 p->key 大于、等于 x，则删除 p->rchild 和 p，

（3） 左走一步，转（2）

（4） 如果 p->key 小于 x，则反复右走，

（5） 转（2）

（6） 直到 p==NULL

8.15 在平衡二叉排序树的每个结点中增加一个 lsize 域，其值为它的左子树中的结点数加 1。编写一时间复杂度为 O(log n)的算法，确定树中第 k 个结点的位置。

[提示]：先画图手工求。

（1）sum=0,

（2）从当前根开始沿左链找 sum + lsize<=k 的最大结点 a，

（3）沿 a 的右链求 sum=sum + lsize，直到结点 b，sum + lsize(b)>=k

重复（2）、（3），直到 sum=k

第九章

9.1 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（前三趟）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。

9.2 一组关键字码，40，27，28，12，15，50，7，采用快速排序或堆排序，写出每趟排序结果。

9.3 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。

n=7 时在最好情况下需进行多少次比较？请说明理由。

对 n=7 给出一个最好情况的初始排列实例。

（保证每个基准记录都是中间记录）

9.4 假设序列由 n 个关键字不同的记录构成，要求不经排序而从中选出关键字从大到小顺序的前 k(k<<n)个记录。试问如何进行才能使所作的关键字间比较次数达到最小？

（简单选择、冒泡排序、堆排序均有可能）

9.5 插入排序中找插入位置的操作可以通过二分查找的方法来实现。试据此写一个改进后的插入排序算法。

9.6 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。

[提示]：（1）参快速排序（2）“水底”、“水面”相遇时结束。

9.7 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：

采取顺序存储结构，至多使用一个记录的辅助存储空间；

算法的时间复杂度 $O(n)$；

讨论算法中记录的最大移动次数。

[提示]：r[0]=r[1]，以 0 为分界值，参快速排序划分过程，但不要求对元素排序。

9.8 试以单链表为存储结构实现简单选择排序的算法

9.9 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。则可按如下方法排序：另设数组 number[v...w]且令 number[i]为统计关键字取整数 I 的记录数，之后按 number 重排序列以达到有序，编写算法实现上述排序方法，并讨论此方法的优缺点。

9.10 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= floor (sqrt (n)). 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。

9.11 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；第一趟对所有奇数 i，第二趟对所有偶数 i，…,依次类推直至整个序列有序为止。

（1） 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。

（3） 写出奇偶交换排序的算法。

9.12 设计一个用链表表示的直接选择排序算法。（与 9.8 重）

9.13 插入排序中找插入位置的操作可以通过二分查找的方法来实现。试据此写一个改进后的插入排序算法。
（与 9.5 重复）

9.14 一个线性表中的元素为正整数或负整数。设计一个算法，将正整数和负整数分开，使线性表的前一半为负整数，后一半为正整数。不要求对元素排序，但要尽量减少交换次数。
（与 9.7 类似）

9.15 为什么通常使用一维数组作为堆的存放形式？

9.16 已知（k_1,k_2,…，k_n）是堆，写一个算法将（k_1,k_2,…，k_n，k_{n+1}）调整为堆。按此思想写一个从空堆开始一个一个添入元素的建堆算法。

9.17 试比较直接插入排序、简单选择排序、快速排序、堆排序、归并排序、希尔排序和基数排序的时空性能、稳定性和适用情况。

9.18 在供选择的答案中填入正确答案：

1)、排序（分类）的方法有许多种：__A ③_法从未排序序列中依次取出元素，与排序序列（初始为空）中的元素作比较，将其放入已排序列的正确位置上；__B①__法从未排序序列中挑选元素，并将其依次放入已排序序（初始时为空）的一端；交换排序法是对序列中元素进行一系列的比较，当被比较的两元素逆序时进行交换。__C ④__和__D②__是基于这类方法的两种排序方法，而__D__是比__C__效率更高的方法，利用某种算法，根据元素的关键值计算出排序位置的方法是__E ⑦_。

供选择答案

① 选择排序 ② 快速排序 ③ 插入排序 ④ 冒泡排序
⑤ 归并排序 ⑥ 二分排序 ⑦ 哈希排序 ⑧ 基数排序

2)、一组记录的关键字为（46，79，56，38，40，84），利用快速排序的方法，以第一个记录为基准得到的一次划分结果为__C____。

A、38，40，46，56，79，84
B、40，38，46，79，56，84
C、40，38，46，56，79，84
D、40，38，46，84，56，79

3)、下列排序算法中，__C__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、SHELL 排序

9.19　判断正误：

（　　）在一个大堆中，最小元素不一定在最后。

（ X ）对n个记录采用快速排序方法进行排序，最坏情况下所需时间是o（$n\log_2 n$）。

（ X ）在执行某排序算法过程中，出现了排序码朝着与最终排序序列相反方向移动的现象，则称该算法是不稳定的。

http://jpkc.nwu.edu.cn/sjjg/pick_title/exercises/hints.htm

第 1 章 绪论

1．4 试编写算法，求一元多项式$P_n(x)=a_0+a_1x+a_2x^2+a_3x^3+\ldots a_nx^n$的值$P_n(x_0)$，并确定算法中的每一语句的执行次数和整个算法的时间复杂度，要求时间复杂度尽可能小，规定算法中不能使用求幂函数。注意：本题中的输入$a_i(i=0,1,\ldots,n)$,x和n，输出为$P_n(x_0)$。通常算法的输入和输出可采用下列两种方式之一：

（1）通过参数表中的参数显式传递。

（2）通过全局变量隐式传递。

试讨论这两种方法的优缺点，并在本题算法中以你认为较好的一种方式实现输入和输出。

```
void polyvalue()
{
int n,p,i,x,xp,sum;
 float a[];
 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
```

第二章 线性表

2．4 设线性表存于 a(1:arrsize)的前 elenum 个分量中且递增有序。试写一算法，将 X 插入到线性表的适当位置上，以保持线性表的有序性。

```
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
```

2．6 已知线性表中的元素（整数）以值递增有序排列，并以单链表作存储结构。试写一高效算法，删除表中所有大于 mink 且小于 maxk 的元素（若表中存在这样的元素），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，它们的值为任意的整数）。

```
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) q=q->next; //q 是第一个不小于 maxk 的元素
  p->next=q;
 }
}//Delete_Between
```

2．7 试分别以不同的存储结构实现线性表的就地逆置算法，即在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）。

（1） 以一维数组作存储结构，设线性表存于 a(1:arrsize)的前 elenum 个分量中。

（2）以单链表作存储结构。

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

2.8 假设两个按元素值递增有序排列的线性表 A 和 B，均以单链表作为存储结构，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）结点空间存放表 C。

```
 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
```

分析:本算法的思想是,按从小到大的顺序依次把 A 和 B 的元素插入新表的头部 pc 处,最后处理 A 或 B 的剩余元素.

2．9 假设有一个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 所指结点的前趋结点。

```
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
```

2．10 已知有单链表表示的线性表中含有三类字符的数据元素（如字母字符、数字字符和其它字符），试编写算法来构造三个以循环链表表示的线性表，使每个表中只含同一类的字符，且利用原表中的结点空间作为这三个表的结点空间，头结点可另辟空间。

```
Status LinkList_Divide(LinkList &L,CiList &A,CiList &B,CiList &C)//把单链表 L 的元素按类型分为三个循环链表.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
```

2．11 设线性表$A=(a_1, a_2,\ldots,a_m)$，$B=(b_1, b_2,\ldots,b_n)$，试写一个按下列规则合并A、B为线性表C的算法，使得：

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

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

线性表 A、B、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。

```
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
```

2．12 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来构成这两个链表。

```
void Divide_LinkedPoly(LinkedPoly &L,&A,&B)//把循环链表存储的稀疏多项式 L 拆成只含奇次项的 A 和只含偶次项的 B
{
 p=L->next;
 A=(PolyNode*)malloc(sizeof(PolyNode));
 B=(PolyNode*)malloc(sizeof(PolyNode));
 pa=A;pb=B;
 while(p!=L)
 {
  if(p->data.exp!=2*(p->data.exp/2))
  {
   pa->next=p;pa=p;
  }
  else
  {
   pb->next=p;pb=p;
  }
  p=p->next;
 }//while
 pa->next=A;pb->next=B;
}//Divide_LinkedPoly
```

2．14 设多项式 P(x)采用课本中所述链接方法存储。写一算法，对给定的 x 值，求 P(x)的值。

```
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
```

第 3 章 限定性线性表——栈和队列

3．5 试写一个算法，判断依次读入的一个以@为结束符的字母序列，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。

```
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
```

3．6 假设表达式由单字母变量和双目四则运算算符构成。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。

```
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 //参见编译原理教材
```

3．7 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、入队列和出队列的算法。

```
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
```

3.8 要求循环队列不损失一个空间全部都能得到利用，设置一个标志域 tag，以 tag 为 0 或 1 来区分头尾指针相同时的队列状态的空与满，请编写与此结构相应的入队与出队算法。

```
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
```

分析:当循环队列容量较小而队列中每个元素占的空间较多时,此种表示方法可以节约较多的存储空间,较有价值.

第 4 章 串

4．2 编写算法，实现串的基本操作 StrReplace(S,T,V)。

StrCat(S,T)； SubString(Sub,S,pos,len)。

```
int String_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<=S[0]-T[0]+1;i++)
 {
  for(j=i,k=1;T[k]&&S[j]==T[k];j++,k++);
  if(k>T[0]) //找到了与 T 匹配的子串:分三种情况处理
  {
   if(T[0]==V[0])
    for(l=1;l<=T[0];l++) //新子串长度与原子串相同时:直接替换
     S[i+l-1]=V[l];
   else if(T[0]<V[0]) //新子串长度大于原子串时:先将后部右移
   {
    for(l=S[0];l>=i+T[0];l--)
     S[l+V[0]-T[0]]=S[l];
    for(l=1;l<=V[0];l++)
     S[i+l-1]=V[l];
   }
   else //新子串长度小于原子串时:先将后部左移
   {
    for(l=i+V[0];l<=S[0]+V[0]-T[0];l++)
     S[l]=S[l-V[0]+T[0]];
```

```
      for(l=1;l<=V[0];l++)
        S[i+l-1]=V[l];
     }
     S[0]=S[0]-T[0]+V[0];
     i+=V[0];n++;
   }//if
  }//for
  return n;
}//String_Replace
```

4．3 假设以块链结构表示串，块的大小为 1，且附设头结点。
试编写算法，实现串的下列基本操作：

StrAsign(S,chars)； StrCopy(S,T)； StrCompare(S,T)； StrLength(S)；

```
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 业已存在.
{
 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
```

4．10 写算法，实现顺序串的基本操作 StrCompare(s, t)。

```
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
```

4．11 写算法，实现顺序串的基本操作 StrReplace(&s, t, v)。

```
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
```

实习题

1．　一、已知串 S 和 T，试以以下两种方式编写算法，求得所有包含在 S 中而不包

含在 T 中的字符构成的新串 R，以及新串 R 中每个字符在串 S 中第一次出现的位置。

（1） 利用 CONCAT、LEN、SUB 和 EQUAL 四种基本运算来实现。

（2）以顺序串作为存储结构来实现。

```
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
```

第 5 章 数组和广义表

5．3 假设稀疏矩阵 A 和 B 均以三元组表作为存储结构。试写出矩阵相加的算法，另设三元组表 C 存放结果矩阵。

```
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
```

实习题

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，试编写算法求出矩阵中的所有马鞍点。

```
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(flag)
       printf("Found a saddle element!\nA[%d][%d]=%d",i,j,A[i][j]);
    }
  }//for
}//Get_Saddle
```

第 6 章 树和二叉树

6．12 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。

```
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
```

6．13 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。

```
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
```

6．14 分别写函数完成：在先序线索二叉树 T 中，查找给定结点*p 在先序序列中的后继。在后序线索二叉树 T 中，查找给定结点*p 在后序序列中的前驱。

```
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
```

分析:总觉得不会这么简单.是不是哪儿理解错了?

```
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;
 }//else
}//PostOrder_Next
```

6．16 编写算法，对一棵以孩子-兄弟链表表示的树统计其叶子的个数。

```
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
```

6．17 对以孩子-兄弟链表表示的树编写计算树的深度的算法。

```
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
```

6．18 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后序遍历非递归的算法。

```
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
```

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.

6．21 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。

```
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
```

6．24 二叉树按照二叉链表方式存储，编写算法，将二叉树左右子树进行交换。

```
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
```

第 7 章 图

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。

```
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;
  }//while
  return OK;
}//Build_AdjList
```

7．6 试在邻接矩阵存储结构上实现图的基本操作：InsertVertex(G,v)，InsertArc(G, v, w)，DeleteVertex(G,v)和 DeleteArc(G, v, w)。

```
//本题中的图 G 均为有向无权图,其余情况容易由此写出
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
```

分析:如果不把待删除顶点交换到最后一个顶点的话,算法将会比较复杂,而伴随着大量元素的移动,时间复杂度也会大大增加.

```
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
```

7．7 试用邻接表存储结构重做题 7.6。

```
//为节省篇幅,本题只给出 Insert_Arc 算法.其余算法请自行写出.

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;q=q->nextarc)
      if(q->adjvex==j) return ERROR; //边已经存在
    q->nextarc=p;
  }
  G.arcnum++;
  return OK;
}//Insert_Arc
```

7．8 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≠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。

```
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
```

7．9 同上题要求。试基于图的广度优先搜索策略写一算法。

```
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
```

7．10 试利用栈的基本操作，编写按深度优先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 看成是一种抽象数据类型。

```
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
```

分析:本算法的基本思想与二叉树的先序遍历非递归算法相同,请参考 6.37.由于是强连通图,所以从第一个结点出发一定能够访问到所有结点.

7．11 采用邻接表存储结构，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（指顶点序列中不含有重现的顶点）的算法。

```
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
```

7．13 已知一棵以二叉链表作存储结构的二叉树，试编写按层次顺序（同一层自左至右）遍历二叉树的算法。

```
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
```

第 8 章 查找

8．8 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。

```
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
```

8．14 写一时间复杂度为$O(\log_2 n+m)$的算法，删除二叉排序树中所有关键字不小于x的结点，并释放结点空间。其中n为树中的结点个数，m为被删除的结点个数。

```
void Delete_NLT(BiTree &T,int x)//删除二叉排序树 T 中所有不小于 x 元素结点,并释放空间
{
 if(T->rchild) Delete_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
 q=T;
 T=T->lchild;
 free(q); //如果树根不小于 x,则删除树根,并以左子树的根作为新的树根
 if(T) Delete_NLT(T,x); //继续在左子树中执行算法
}//Delete_NLT
```

8．15 在平衡二叉排序树的每个结点中增加一个 lsize 域，其值为它的左子树中的结点数加 1。编写一时间复杂度为 O(log n)的算法，确定数中第 k 个结点的位置。

```
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
```

第 9 章 内部排序

9．6 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。

```
void Bubble_Sort2(int a[ ],int n)//相邻两趟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
```

9．7 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：
采取顺序存储结构，至多使用一个记录的辅助存储空间；
算法的时间复杂度 $O(n)$；

讨论算法中记录的最大移动次数。

```
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
```

9．8 试以单链表为存储结构实现简单选择排序的算法

```
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
```

9．9 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。则可按如下方法排序：另设数组 number[v...w] 且令 number[i]为统计关键字取整数 I 的记录数，之后按 number 重排序列以达到有序，编写算法实现上述排序方法，并讨论此方法的优缺点。

```
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
```

分析:本算法参考了第五章三元组稀疏矩阵转置的算法思想,其中的 pos 数组和那里的 cpot 数组起的是相类似的作用.

9．10 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=?√n?.　　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。

```
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,算法原理参见 5.18
{
 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
```

9．12 设计一个用链表表示的直接选择排序算法。

```
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
```

9．16 已知（$k_1, k_2, \cdots, k_n$）是堆，写一个算法将（$k_1, k_2, \cdots, k_n, k_{n+1}$）调整为堆。按此思想写一个从空堆开始一个一个添入元素的建堆算法。

```
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
```