第一章

一、主要术语和概念

1、信息：是向人类（或机器）提供的关于现实世界中有关事务的知识或者信息

2、信息处理：采用计算机对数据进行加工处理，包括对数据进行的接受及输入、存储、转化、发布等的处理，将信息变成客观存在的过程

3、数据：用来描述客观实体，并能被计算机识别的符号集

4、数据元素：通常具有完整确定意义的，描述现实世界某客观实体的最小数据集

5、数据项：构成数据元素的若干个不可再分的数据要素，是数据不可分割的最小单位

6、数据对象：具有相同属性的数据元素的集合

7、数据结构：给定数据对象和对其定义的操作所共同构成的系统

二、本章主要内容

1、逻辑关系、结构：对数据对象中的数据元素之间的一种联系和对它们之间关系的描述。分为线性关系和非线性关系（又分为树形关系和图关系）

2、存储关系：是对逻辑关系的在存储器中的实现。

顺序存储：在存储区域按连续地址空间顺序依次存放数据元素

链接存储：将结点所占的存储单元分为两部分，一部分存放结点本身信息，另一部分存放此结点关联的其他结点的信息

索引存储：将数据的存储分为两部分，一个为数据区，所有存储结点连续存放；索引区中的索引指出各个存储结点的存储位置

散列存储：根据结点的值，通过一定的函数关系来确定数据元素的存储结构

3、运算关系：定义在逻辑关系但需要依靠存储关系实现

4、数据结构主要通过算法的时间效率和存储需要量来评价

第二章 算法概述

一、主要术语和概念

1、算法：是解决某个问题的指令的有限集合，都必须符合有穷性、确定性（每一步没有二义性）、可行性（每条指令是来自于指令集），并有输入和输出

2、过程：事务发展或事物进行所经过的程序，基于其目的可分为一般过程，注重过程；函数过程，则注重结果

3、程序：是算法在计算机中用程序设计语言的实现

4、时间 \ 空间复杂性：利用某算法处理一个问题规模为n的输入所需要的时间 \ 存储空间

5、难解性问题：不存在以多项式时间为界限的算法的问题

6、静态链表：以整型变量的值作为存储链接的地址值来联系结点的整体。

二、主要内容

1、算法设计：遵循的一般准则为正确性（能得到正确的结果）、易读性（为方便交流）、健壮性（对不当的输入的反应不导致自身崩溃）和高效性（高效率、低存储量）。自顶向下、逐步求精的方法来设计

2、算法的评价：从时间和空间复杂性评价，具体讨论又从最坏情况和平均情况讨论。求时间复杂度，先选择一种或几种操作作为“标准操作“，再确定算法在给定输入情况下共执行了标准算法的次数，由此导出时间函数；比较空间复杂度则是比较在相同输入数据规模和存储结构的情况下，算法所需的附加存储空间。

3、算法的描述：常见描述为PDL语言（伪码语言）、流程图、自然语言和程序设计语言。算法的书写主要有一般结构：

PROCEDURE 过程名（I\O参数）

BEGIN

语句组

END；

和函数过程：

FUNCTION 函数名（I\O参数）：类型名；

BEGIN

语句组

END；

基本语句有：赋值语句、条件语句（IF THEN ）、循环语句（WHILE、 REPEAT UNTIL、FOR TO\DOWNTO）和输入输出语句（read()、 write() ）

第三章 线性表

一、主要术语和概念

1、线性表：是n>=0个数据元素a1, a2, …, an的有限序列，序列中除第一个和最后一个外，每个元素有且仅有一个直接前驱和直接后继。表中的元素呈线性关系

2、线性表的长度：表中元素的个数，空表长度为0

3、记录、文件：当一个数据元素由若干个数据项组成，一个数据元素可称为记录，线性表称文件

4、向量：顺序地分配存储单元，且每个数据元素占用相同大小的存储空间

5、链表：通过指针联系起来的结点的整体

二、主要模型和算法

1、利用向量存储

设A = (a1, a2, …, an)，每个元素占用m个单元，则Loc(ai) = Loc(ai – 1) + m = Loc(a1) + m(i - 1)。因为连续分配的且具有线性关系，可用一位数组实现。

插入运算：在表A = (a1, a2, …, an) 的第i – 1和第i个元素之间插入一个新的元素b。属于一般过程，常规实现过程为将第i个元素及其之后的元素向后移动一个单元（利用循环语句），再将第i个单元赋值为b。对于i < 1或i > n + 1，n = max的边界情况，作为例外的条件加入到常规情况当中即可。即if ( i < 1 or i > n + 1) then exit; if ( n = max) then write (‘no insert’);

2、利用链表存储

利用指针表示结点之间的关系而非存储位置，一个结点用数据域存储数据元素的信息，指针域存储其直接后继的存储位置

插入运算：将数据元素b插入到单链表H中的第一个数据元素为a的结点之前。常规过程为，利用单链表，为数据元素b申请空间，并完成数据存储，作为结点S；从头指针开始在H中搜索第一个数据元素为a的结点，a前结点的指针域指向b，b指针域指向a完成插入。考虑异常情况 (1) H = nil时，直接将S作为链表最后一个结点 (2) 第一个结点为a时，直接将S作为链表第一个结点。最后将异常情况和平凡情况合并即可。

对算法的优化

1. 对单链表加入表头：可将第一个结点是a的情况合并到平凡情况中
2. 将带表头的单链表改为带表头的循环单链表：又可将链表为空的情况划为平凡情况，此时，所有情况都可划归平凡情况。

PROC InsLinkList(VAR H:link; a, b:datatype);

{把b插入到循环单链表H的第一个a之前}

BEGIN NEW(S); S↑.data := b;

H↑.data := a; SAVE := H; P := H↑.next;

WHILE P↑.data != a DO [

SAVE := P; P := P↑.next ]

S↑.next := p; SAVE↑.next := S;

END;

1. 再更改为带表头结点的双向循环链表，可弥补单链表只能单向行进的缺点，可实现双向查找。

删除运算：在单链表H中删除第一个数据元素为a的结点。但在访问到a后，无法知道a前面的结点，因此引入指针SAVE来保留。同时，在删除后，回收结点。

3、静态链表

存在该种形式是因为一些高级语言无法实现动态链表仿真的动态过程

三、主要算法

1、多项式的求解问题

多项式的表示可用单链表实现，对相加和相减，每个多项式设置一个指针。以一个多项式A为基准，和另一个多项式B比较，若指数不同，两者指针向前移动，相同，对系数进行运算，结果放入新设立存储结果的链表C中。当一个多项式处理结束，另一个还有剩余项，直接复制到C中。

第四章 栈和队列

一、主要术语和概念

1、栈：是一个下限为常数，上限可变化的向量（或反之）。或定义为，运算是线性表的子集，且子集还要加以限制

2、栈顶、栈底：栈中可变化的一段称为栈顶，不变化的是栈底。

3、入栈、出栈：完成向栈中加入数据元素的过程即入栈，完成向栈中取出数据元素的过程即出栈

4、桟基：在栈顶到栈底的方向上，再后退一格，将上一个栈的最顶部作为下一个栈的桟基。

5、递归：即一个过程(或函数、数据结构)问题的定义(解决)中又应用了它自身(作为定义项之一),那么该过程(或函数、数据结构)称为是递归定义的

6、队列：是一个下限和上限只能增加而不能减少的向量

7、队首、队尾：取出元素的一端称队首，加入元素的一端称队尾。

二、主要模型

1、栈的存储

利用向量表示栈。入栈过程，将数据元素存入栈顶所在位置，栈顶加一，出栈为其逆过程。或利用链表链式存储。

2、多栈共存问题

分为两个栈共存和n个栈共存，

两个栈共存：将一个栈分为两部分，两个栈顶相邻，栈顶在最两端，以解决假溢出的问题。

n(n > 2)个栈共存：以两个栈为单位，每个单元按照两个栈共存的问题解决。每个栈的栈顶和栈底指针统一分类别保存。 i) 存在弹出判别栈是否为空的条件bottom(i) = top(i) 和初始时bottom(i) = top(i) 的条件冲突，因此，引入栈基base( )。此时，栈空判断条件为：base(i) = top(i)，初始化：base(i) = (i - 1) × ( [ m/n ] )（n为栈的个数，m为单个栈中的个数）。 ii) 压栈时栈满如何判别，在第1到第n – 1个栈：top(i) = base(i + 1)，第n个栈top(i) = m。为提高效率，为第n个栈虚设一个桟基，即再增加一个桟基。 iii）第i个栈假溢出问题，处理原则：从右到左，寻找最近未满的栈，通过左右移动元素产生空位。

1. 估计n个栈所占空间容量(空间)
2. 如果sum=m则溢出停机。
3. 从第i+1个栈到第n个栈寻找未满的栈,若有转④,否则转⑤。
4. 设第 j 个栈( i+1 ≤ j ≤ n )未满，则将栈中从top(i)+1序号到top(j)之间的分量右移一个位置，并修改相应栈基栈顶指针，转⑦。
5. 从第 i-1 个栈到第 1 个栈寻找未满的栈。
6. 设第j 个栈( 1 ≤ j ≤ i-1 )未满,将栈中从base(j+1)+1序号到top(i)之间的分量左移一个位置,并修改相应栈基栈顶指针。
7. 将要入栈的元素送入第top(i)+1个单元，该对应栈顶指针加1(top(i)+1top(i)) 。

3、队列的运算：入队，完成往队列中加入数据元素的过程，即插入操作；出队，完成从队列中取出数据元素的过程,即删除操作。

4、队列的存储：i）链式存储（链表）ii）向量存储，缺点：插入和删除需要移动的元素很多，改进：增加收尾指针，并牺牲一个单元，队首即front + 1。对假溢出问题的解决采用循环队列。因此，入队出队过程课改进为：

PROC AddQ(CQ, x, front, rear)

BEGIN rear←rear+1;

IF rear=n+1 THEN rear←m;

IF rear=front THEN write(“queue full”)

ELSE CQ[rear]←x

END;

PROC DelQ(CQ, x, front, rear)

BEGIN

IF rear=front THEN write(“queue empty”)

ELSE IF front=n THEN front←m

ELSE front←front+1

END;

5、双端队列：是一种线性表,对它所有的插入和删除都限制在表的两端进行。

6、双栈：是一种加限制的双端队列,它规定从end1插入的数据元素只能从end1端删除,而从end2端插入的数据元素只能从end2端删除。与两个底部相连的栈相似。

7、超队列：是一种删除受限制的双端队列,即删除限制在一端(例如end1)进行,而插入仍允许在两端进行。像一种特殊的优先队列，允许最新插入的数据元素，以最高的优先级删除。

8、超栈：是一种插入受限制的双端队列,即插入限制在一端(例如end2)进行,而删除仍允许在两端进行。它可以看成对栈溢出时的一种特殊处理,即当栈溢出时,可以将栈中保存最久(end1端)的数据元素删除。

三、主要算法、

1、表达式的求值问题

借助栈，设立操作数栈和运算符栈，遇操作数，压入操作数栈，遇运算符，判断运算符级别，当运算符级别较高，从操作数栈中取出两个进行运算，运算结果再次压栈。右括号按照最高级别进行运算。

2、递归问题的求解

递归问题需借助栈结构。递归必须有基本项和诱导项。

IF 由问题参数构成的条件 THEN 基本函数

ELSE 诱导函数

第五章 串

一、主要术语和概念

1、串：一个由零个或多个字符组成的有穷序列。

2、串的长度：串中所含的字符个数

3、空串、非空串：串长为零的串称为空串，其它串称为非空串

4、子串、主串：一个串中任意个连续字符组成的子序列称为该串的子串，该串成为它的所有子串的主串。

5、串的模式匹配：两个串P和S，在S中找出一个与P相同的子串，把从目标S

中查找模式P的过程称为串的模式匹配

二、主要模型

1、串的运算：i）赋值：将字符串常量chars赋给字符串变量S ii）联接：将字符串S和T联接在一起 iii）取子串：返回S的一个子串 等等

2、串的存储：i）顺序存储，通常向量表示

TYPE string=ARRAY [1..m] OF char;

VAR S:string;

ii）索引存储 iii）链接存储，使用单链表或循环链表

三、主要算法

1、插入运算：把串T插入到S的第i个字符之后。

PROC StringIns(VAR S,T:link;i:intrger);

BEGIN CASE

i<0 : write('i error')

T=nil : exit;

S=nil : S←T;

ELSE 【 P ←S; j←1

WHILE ( j<i )AND (P.next≠nil ) DO

【 j←j+1; P←P.next 】;

IF i=0 THEN【 save←S ; P←T; S←T 】

ELSE【 save←P.next; P.next←T】;

WHILE (P.next≠nil ) DO P←P.next ;

P.next ← save 】

ENDCASE

END;

2、串的联接运算：串S和T相加，成为串R

PROC Concatenation(VAR S,T,R:string);

BEGIN CASE

ls+lt≤lr : [m ← ls ; n ← ls+lt ]

ls>lr : [m ← lr ; n ← 0 ];

ls<lr 且ls+lt>lr : [m ← ls ; n ← lr ];

ENDCASE

FOR i← 1 TO m DO R[i] ← S[i];

FOR i← m+1 TO n DO R[i] ← T[i-m]

END;

3、串的模式匹配

i）匹配的朴素算法(Brute-Force算法) 对正文顺序搜索，大致步骤为：①逐次比较(对应字符依次比较) ②发现不匹配时，将P相对于S右移一位 ③重复上述过程，直到成功或扫描完S。

PROC Search(VAR S,P:string;i:intrger);

BEGIN i←0;

WHILE i<n DO

【 i←i+1 ; j←i ; k←1;

WHILE S[j]=P[k] DO

IF k=m THEN【 WRITE( i, 'success '); exit 】

ELSE【 j←j+1 ; k←k+1】 】 ;

WRITE('failure')

END;

ii）KMP算法，对朴素算法的改进，当指针移动时，若有不同的字符，则说明没有相同的子串

PROC KMP (VAR S，P:string; i :intrger);

BEGIN

i←1; j←1;

WHILE i≤n DO

【 WHILE (j>0) AND (S[i]≠P[j]) DO j←next[j];

IF j=m THEN 【 WRITE( i, 'success '); exit 】

ELSE【 i←i+1 ; j←j+1】 】;

WRITE('failure')

END;

next的定义为：①0 当j=1 ②max{k∣k,1＜k＜j，使得substr(P，1，k - 1) = substr(j-k+1，j-1)非空 ③1为其余情况时

next( j )函数

PROC KMP\_next\_mend(VAR S:string; VAR next:ARRAY[1..m] OF :intrger);

BEGIN next[1]←0; j←2;

WHILE j≤m DO

【 k←next[j-1];

WHILE (k≠0) AND (P[k]≠P[j-1]) DO k←next[k];

next[j] ←k+1;

j←j+1 】

FOR j=2 TO m DO

IF P[j]=P[next[j]] THEN next[j]←next[next[j]]

END;

第六章 数组和广义表

一、主要术语和概念

1、一维\n维数组：一维数组是个向量，它的每个元素是该结构中不可分割的最小单位；n(n>1)维数组是个向量，它的每个元素是n-1维数组，且具有相同的下限和上限。

2、向量：标量的一维的有序集合

3、数组与向量的关系：物理关系：向量是数组的特例(一维数组就是向量)，且具有复杂的访问公式。逻辑关系：数组是向量的扩充(向量的向量)。

4、稀疏矩阵：在一个数组当中和某元素比较而言,不相同的元素很少时,我们称此数组为稀疏数组。（某元素：为了讨论方便一般可以认为是零，很少：个数远远小于,例，2维数组中p＜＜m×n）

5、广义表：是零个或多个原子或子表所组成的有限序列，A=(a1,a2,…,ai ,…,an)，ai称为原子，做表中的构成元素的表称为子表

二、主要模型

1、数组的运算：给定一组下标，存取相应数据元素。给定一组下标,修改相应数据元素的某个数据项的值。

2、数组元素的地址访问：i）二维数组(行优先)地址访问：设数组A(c1..d1,c2..d2)，首地址为Loc(c1,c2)=AO(A),元素长度l。Loc (i, j) = AO(A)+l\*[(i-c1)(d2-c2+1)+(j-c2)] ii）多维数组地址访问(行优先)：设数组A(c1..d1, c2..d2, …, cn..dn)，首地址为Loc(c1,c2 ,c3 ,… ,cn) = AO(A), 元素长度l。Loc(j1，j2，…，jn) = AO(A)+l (j1-c1)(d2-c2+1)(d3-c3+1)… (dn-cn+1)+(j2-c2)(d3-c3+1) … (dn-cn+1)+…+(jn-1-cn-1)(dn-cn+1)+ (jn-cn) ＝ AO(A)+l∑(ji-ci)∏(dk-ck+1) + (jn-cn)  虚拟地址：VO(A)＝AO(A)－∑ici iii）对特殊数组，三角矩阵：Loc(aij) = Loc(a11)+[i (i-1)/2+(j-1)]×l 1≤j≤i≤n，三对角矩阵，Loc(aij) = Loc(a11)+[ 2(i-1)+j-1]×l（i=1, j=1, 2 或 i=n, j=n-1,n 或1<i<n,j=i-1,i,i+1）

3、稀疏矩阵：与特殊矩阵的差别：零元素分布有无规律。

4、稀疏矩阵的存储：i）顺序存储，行(或列)优先的原则 ,将矩阵中的非零元素顺序存放,为便于检索和存取,一般带有辅助信息。(1).三元组，将矩阵的全部非零元素抽象为一个线性表，每个元素含行、列、值三个数据项。①抽象为由记录构成的一维数组 ②抽象成具有3列的一个2维数组，A(0..p, 1..3)，其中p为非零元个数 (2).索引表示(二元组表示)，对三元组中的row（若有序），建立行索引表

索引表向量为 pot[1]=1

pot[i]=pot[i-1]+num[i-1]，当2≤i≤n+1 其中num[j]为第j行非零元个数,1≤j≤n

检索步骤为：1.先在索引表中定位主表位置,第i行(1≤i≤n)首元素位置值：

IF pot(i+1)-pot(i)=0 THEN 第i行无非零元素

ELSE pot(i)为第i行首元素位置

2.在主表中顺序搜索,或二分搜索

（3）伪地址表示，按照非零元在矩阵中出现的相对次序作为元素的地址映射关系来组织元素次序(在行或列优先的次序下)。

ii）链接存储，（1）单链表表示，将矩阵每行(行优先次序下)非零元素组织在一个单链表中,链表结点保留列号和元素值。（2）十字链表表示，对于非零元素依据行、列特性,每一行和每一列建立一个循环链表。

iii）散列存储，以元素的行、列下标值作为参数构造散列函数

5、广义表的存储，利用顺序或链表方式，对链表，结点有三个数据域，atom、info、link，atom区分结点是子表还是原子，atom = 0，结点为子表，info存放该子表首元素地址；atom = 1，结点为原子，info存放信息或地址；link存放同层下一元素地址

第七章 树形结构

一、主要术语和概念

1、树：T, 是满足如下性质的有限个结点组成的非空集合：①T中有且仅有一个称为根的结点；②除根结点之外，其余结点分成m(m>0)个不相交的集合T1,T2,…,Tm,其中每个Ti都是树，而且都称为T的子树。

2、有序树：在树T中如果子树的相对次序是有序，则T为有序树

3、森林：是零棵或多棵不相交的树的集合(通常有序)

4、结点的度：子树的个数，出度

5、树的高度：树中结点的最大层次数

6、二叉树：T，是满足如下性质的结点的有限集合。T是空集，或者T包含一个根结点且其余结点分成两个不相交的集合，并分别被称为根结点的左子树和右子树

7、遍历：对于给定的数据结构,系统的访问该结构中的每个结点,且每个结点仅被访问一次的操作过程

8、二叉树的遍历：对于给定的二叉树，系统地访问二叉树中的每个结点，且每个结点仅被访问一次的操作过程

9、线索树：将二叉树的空指针利用起来，用于表示某线性关系下前驱或后继时，这种指针称为线索。带线索的二叉树称为线索二叉树

10、二叉排序树：二叉排序树或者是空二叉树，或者是满足以下性质的二叉树 i）若它的左子树非空，则左子树上所有结点的值均小于根结点的值 ii）若它的右子树非空，则右子树上所有结点的值均大于根结点的值

二、主要模型

1、树的存储：i）双亲表示法，以一组连续的空间存储树的结点，同时在每个结点中附加一个指示器用以指出其双亲的位置 ii）孩子表示法，可采用定长（取子树个数的相对最大值作为子树项的个数）或变长（依据实际子树个数确定子树项个数）结点 iii）孩子兄弟表示法，采用双向链表，左指针存储第一颗子数，中间的data存储当前结点，右指针存储兄弟相邻结点。

2、二叉树的存储：i）顺序存储，利用一维数组，按照每行的顺序存储，但对于非完全二叉树，因为建立存储空间是按照完全二叉树建立，可能会存在大量的空间浪费。 ii）链表表示，左指针指向左子树，右指针指向右子树。

3、线索树：空的左指针域用于表示直接前驱，空的右指针域用于表示直接后继。增加tag，tag = 0时，表示该指针指向孩子；tag = 1时，表示该指针为线索，指向前驱或后继。因为根结点是非平凡情况，因此，增加表头结点进行优化。

4、霍夫曼树(最优二叉树)：实现非等长度下的最优编码，路径长度最短的二叉树

三、主要算法

1、二叉树的遍历次序：先序、中序、后序

先序：

PROC Preorder(VAR T: BinaryTree);

BEGIN IF T≠nil 、

THEN 【 WRITE(T.data)；Call Preorder(T.lson)；Call Preorder(T.rson);】

END;

中序：

PROC Inorder(VAR T: BinaryTree);

BEGIN IF T≠nil

THEN 【 Call Inorder(T.lson)；write(T.data)；Call Inorder(T.rson);】

END；

改变为非递归过程，使用堆栈 (1) 初始化：① 搜索指针授权： p←T ② 栈的初始化：i←0 （2）重复 ①搜索指针P不空：保存当前P；P沿左孩子方向下降 ② S中有东西时：取一个结点数据；访问该结点；控制权移交其右子树

PROC Inorder1(VAR T: BinaryTree);

VAR S: ARRAY[1..max] OF BinaryTree;

BEGIN i←0; {i用作栈顶指针}

p←T; {p用作搜索指针}

REPEAT

WHILE p≠nil DO {遍历左子树}

【 i←i+1; S[i]←p; {入栈}； p←p.lson】；

IF i≠0

THEN 【 p← S[i]; i←i-1; {退栈}；write(p.data); {访问根}；p←p.rson； {遍历右子树}】

UNTIL (i=0) AND (p=nil)

END;

后序：

PROC Postorder(VAR T: BinaryTree);

BEGIN IF T≠nil THEN 【 Call Postorder (T.lson)；Call Postorder (T.rson)；write(T.data)】

END;

非递归过程：当搜索指针指向某结点，先遍历左子树，将此结点入栈保存，当处理完左子树后，遍历其右子树，该结点再次入栈保存。为区别二次入栈需设立标志位tag，tag = 1，该结点可访问；tag = 0，该结点暂不可访问。

2、树转化为二叉树，约定树为有序树，转换规则为，加线，抹线，调整

3、二叉树还原为树，加线（若某结点x是双亲结点的左孩子,则将该结点的右孩子以及当且仅当连续地沿着此右孩子的右链不断搜索到的所有右孩子都与i的双亲结点连接起来），抹线（抹掉结点与所有右孩子之间的连线），调整

4、森林转换为二叉树，先将每一棵树转为二叉树，再将二叉树链接，依据森林中树的次序,依次将转换得来的二叉树，后一棵作为前一棵二叉树的根结点的右子树。

5、二叉树还原为森林，抹线，还原。

6、树的遍历，按照先序或者中序，或者后序的顺序依次遍历结点

7、森林的遍历，转化为二叉树处理

8、按中序线索化二叉树：设 Pr 为待访问结点P的直接前驱结点

PROC threaded(VAR T: BinaryTree);

VAR S: ARRAY[1..max] OF BinaryTree;

BEGIN i←0; p←T; pr←nil;

REPEAT

WHILE p≠nil DO 【 i←i+1; S[i]←p; p←p↑.lson】；

IF i≠0 THEN【 p← S[i]; i←i-1;

IF pr≠nil

THEN【IF pr↑.rson=nil THEN pr↑.rson := - p;

IF p↑.lson=nil THEN p↑.lson := - pr; 】；

pr ← p;

p ← p↑.rson】；

UNTIL (i=0) AND (p=nil)

END;

9、中序线索树中寻找结点后继：若P右子树为空(P↑.rtag=1)，P↑.rson即为其直接后继。若P右子树非空（ P↑.rtag=0 )，P的直接后继在其右子树的沿着左链搜索到的第一个 P↑.ltag=1 的结点，增加表头结点后，算法为：

PROC INSUC\_Head(VAR P,Q: BinaryTree);

BEGIN {排除空树}

IF p↑.lson=p THEN 【 q←p; exit 】 ；

{查找，后继应当在其右链中}

q←p↑.rson;

IF p.rtag=0 {说明有真正的右子树}

THEN WHILE q↑.ltag=0 DO q←q↑.lson

END;

10、按照中序规律遍历中序线索树：(1)寻找二叉树T全局第一个被访问的结点P (2)访问P (3)寻找P的直接后接结点Q，Q=>P (4)重复(2)-(3)直到P=nil

11、中序线索树中插入结点：将要插入的结点T作为结点S的右孩子，若S右子树为空，此时，S的右链应为线索；若S右子树非空，此时，S的右链应为真正的右孩子

PROC insrson(VAR T,S: BinaryTree);

BEGIN T↑.rson←S↑.rson; T↑.rtag←S↑.rtag； T↑.ltag←1; 　 T↑.lson←S; S↑.rtag←0; S↑.rson←T;

IF T↑.rtag=0 THEN 【CALL INSUC(T,Q); Q↑.lson←T 】

END;

11、若它的右子树非空，则右子树上所有结点的值均大于根结点的值

PROC InsertNode(VAR T,S: BinaryTree);

BEGIN

IF T=nil THEN T←S

ELSE IF (S.data﹤T.data)

THEN call InsertNode(T.lson,S)

ELSE call InsertNode(T.rson,S)

END;

12、霍夫曼算法：（1）给定一组权值集合{w1, w2, …, wn}，据此构成n棵二叉树组成的森林F。（2）将F={T1, T2, …, Tn}按根结点的值由小到大进行排序。（3）取出T1和T2组成一棵二叉树T；再将T插入到F中，并使F依据根结点的值有序。（4）反复（3），直到F = {T}。利用数组和辅助结构（每棵二叉树根结点的值和每棵二叉树根结点在R中的下标值）存储。

PROC Huffman(R,n);

BEGIN

FOR i←1 TO n DO 【A[i].data←R[i].data;A[i].adr←i】 ;

i←0;

WHILE n-i≥2 DO

【 call insort(A,n-i);

i←i+1;

R[n+i].data←A[1].data+A[2].data;

R[n+i].lson←A[1].adr;

R[n+i].rson←A[2].adr;

A[1].data←R[n+i].data; {将生成的二叉树作为A第1元素}

A[1].adr←n+i;

A[2].data←A[n-i+1].data; {添补A第2元素空位}

A[2].adr←A[n-i+1].adr 】

END;

第八章 图结构

一、主要术语和概念

1、图：由n (n≥1)个结点v1,v2,…,vn 构成的数据G称为图,若结点集V={v1,v2,…,vn}上定义的称为后继的关系E是非自反的。表示为G=(V,E)， V称为顶点集，E称为边集

2、有向图：在图G中，若每个关系都是顶点的有序对，则称G为有向图

3、无向图：图G中,如果每个关系都是顶点的无序对，则称G为无向图

4、子图：设图G=(V,E), 如果有图G‘=(V’, E‘) ，且E’属于E, V‘属于V，则称G’为G的子图

5、网：给图的每一条边加一个(非负的)数，这个与图的边相关的数值称为权(weight)。带权的图称为网，带权的连通图称为网络。

6、图的遍历：给出图G和其中的任意一个顶点v0，从v0出发系统地访问G中所有的顶点,且每个顶点仅被访问一次,这一过程称为图的遍历

7、AOV网：有向图G中,顶点表示活动或任务,有向边表示活动或任务之间的优先关系,则此有向图称为顶点表示活动的网络

8、拓扑序列：对于有向图G=(V,E),V中的顶点的线性序列(vi1, vi2, … , vin)，称作一个拓扑序列

9、关键路径：任务计划作业图上的需要时间最长的路径

二、主要模型

1、图的存储

邻接矩阵，无向图：矩阵第i行之和是顶点vi的度；有向图：矩阵第i行之和是顶点vi的出度，矩阵第i列之和是顶点vi的入度

邻接表：每个顶点建立一个链表，无向图：是与vi邻接的所有结点的收集，有向图：是以vi为始点的所有终点的收集

三、主要算法

1、邻接表的建立

PROC BuildAdjlist(VAR A: adj\_list;e,n:integer);

BEGIN

FOR i←1 TO n DO【 A[i].vertex←i; A[i].link←nil】 ;

FOR i←1 TO e DO

【 read(u,v); {读入有向边<u,v>}

new(P); p↑.adjvex←v;

p↑.next←A[u].link; A[u].link←p】

END;

2、图的遍历，设立辅助数组visited[1..n]，visited[v] = 1时，v已经被访问；visited[v] = 0时，v未被访问。有深度优先搜索（DFS）和广度优先搜索（BFS）

深度优先搜索：先访问出发顶点v0，然后选择一个v0邻接到的未被访问过的结点u，再从u开始进行深度搜索

广度优先搜索：访问出发顶点v0，然后访问v0邻接到的所有未被访问过的结点v1 , v2 , … , vt，再依次访问v1, v2, …, vt 邻接到的所有未被访问的结点。

3、Prim算法：V={1,2,…,n}是图G的顶点集合，由一个初值为{v0}的集合U开始，每次生成一条边，逐渐长成一棵具有最小代价的生成树，每一步中都找出一个最小权的边(u, v)，且u∈U, v ∈V-U.

4、克鲁斯卡尔算法：按照权值递增的顺序逐个考虑E中的每条边，若该边连通了在两个不同连通分量中的顶点，则将该边填加到T中。重复，直到T中包含了n-1条边，终止

5、拓扑排序：即寻找拓扑排序的方法，

基于邻接矩阵：(1)S置零，设定顶点的输出的顺序编号的起始值 (2).寻找没有输出号的全零的列u，如果没有，则算法终止。此时S中所有元素都有输出号，拓扑排序完成；否则有环 (3)将输出次序号赋给S[u] (4)把第u行置成全零 (5).输出次序号加1，回到(2)。

基于邻接表：(1)搜索邻接表中入度为零的顶点，并令其进栈。(2).当栈非空时，进行拓扑排序：i）退栈，输出栈顶元素u ii）在邻接表中循环扫描u的直接后继顶点k，将k的入度减1，若此时k的入度为零,则令k入栈。(3)若栈空，输出元素不足n则有环，否则拓扑排序完成。

6、求关键路径：（1）计算Ve(i),i=1,2,…,n（拓扑排序） （2）计算Vl(i),i=1,2,…,n (利用（1）得到的拓扑序列的逆序） （3）计算e(i)及l(i) (利用①、 ②得到Ve,Vl） （4）计算e(i)-l(i), 输出差值为零的活动

求Ve：

(1).初始化：Ve[1..n] ←0 ;{由于求的是最大值，故所有Ve置零}

(2).搜索邻接表中入度为零的顶点，并令其进栈。

(3).当栈非空时，进行拓扑排序：

①退栈，输出栈顶元素u;

②(循环)在邻接表中扫描u的直接后继顶点k，将k的入度减1，若此时k的入度为零,则令k入栈。经由u若能增大Ve(k),则增大，保存u到逆拓扑栈；

(4).若栈空，输出元素不足n则有环，否则拓扑排序完成。

求Vl：

(1).初始化:若n为工程的终点，不影响工期则：Vl[n]=Ve[n],故：Vl[1..n] ←Ve[n] ; {由于求的是最小值，故所有Vl置最大值}

(2).当逆拓扑栈非空时，进行逆拓扑排序：

①退栈，输出栈顶元素u;

②(循环)在邻接表中扫描u的直接后继顶点k，经由k若能缩小Vl(u),则缩小；

7、迪杰斯特拉算法：存储结构，利用邻接矩阵，最短路径的长度值存储辅助向量dist[ i ]，最短路径的顶点序列存储辅助向量path[ i ]。

（1）初始计算dist[i]，dist[i] = wi，当存在v0到vi的边；dist[i] = 正无穷，其余情况。path初始状态，IF＜v0,vi＞属于E(G) THEN Path[ i ] := {v0,vi} ELSE Path[ i ] := 空集

（2）计算 i）在V-S中挑选最小的dist[u] ，即选择距S最近的顶点u ii）将u并入S iii）通过u尝试缩小V-S中的顶点i的dist[i]，若能缩小则缩小（并同步替换path[i]）