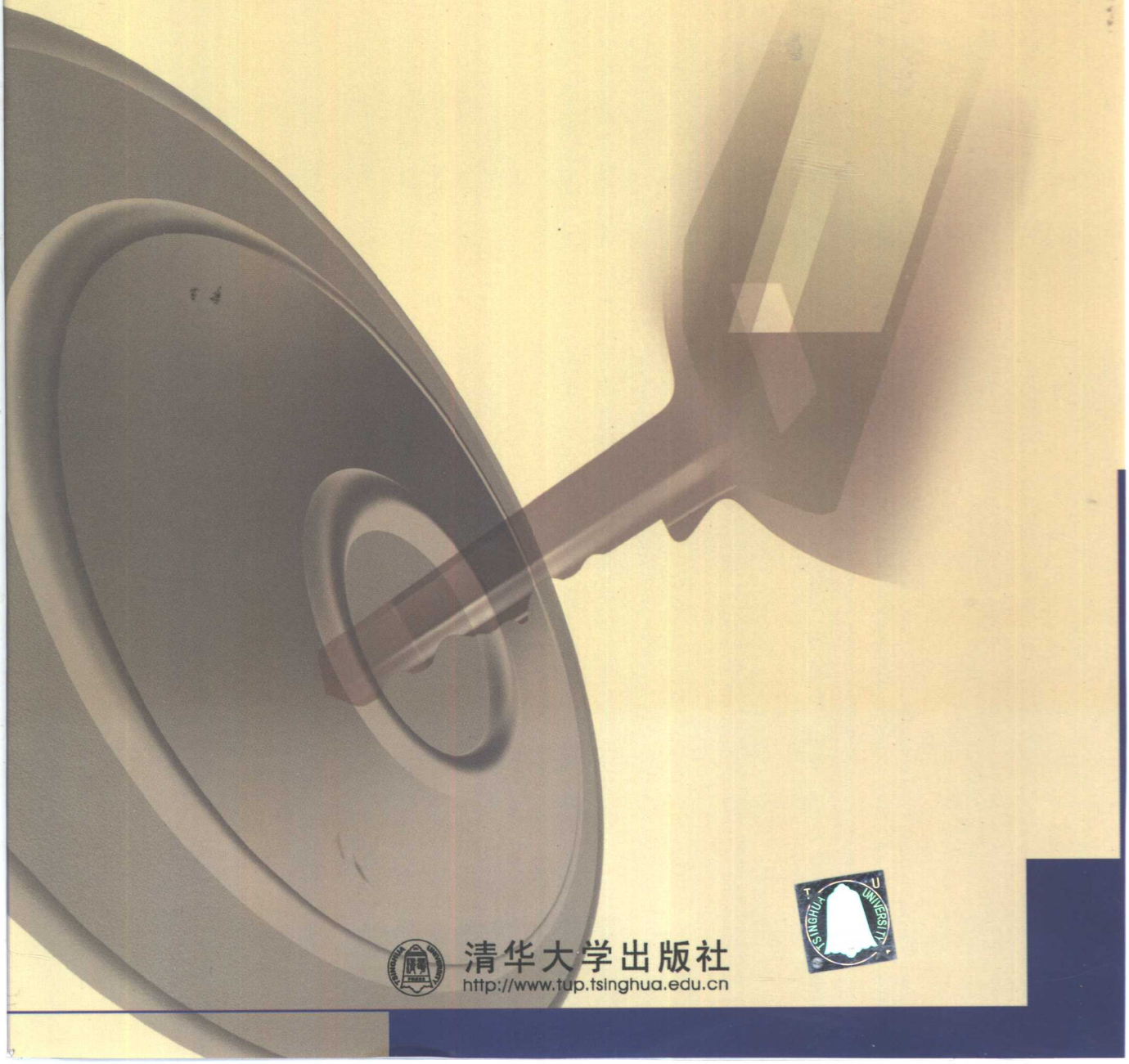
高等教育自学考试

计算机及应用专业（专科）自学辅导丛书

数据结构导论自学考试指导

徐孝凯 编著



清华大学出版社
http://www.tup.tsinghua.edu.cn

高等教育自学考试计算机及应用专业(专科)自学辅导丛书

数据结构导论
自学考试指导

徐孝凯　编著

清华大学出版社

(京)新登字 158 号

内 容 简 介

本书是专门为全国高等教育自学考试计算机及应用专业(专科)"数据结构导论"课程而编写的一本自学指导教材，是严格按照该课程自学考试大纲并配合陈小平主编的《数据结构导论》教材编写的。本书对该课程的所有知识点按章进行了系统的归纳和总结，按章进行了重难点辅导并给出了丰富的练习题及参考解答，练习题型完全与实际考试题型相吻合。附录一给出了指定教材中每章习题的全部参考解答，附录二给出了两套模拟试卷及解答，附录三给出了最近的自考试卷及解答。学习此书将会帮助你比较轻松地学习好"数据结构导论"课程。

书　　名：数据结构导论自学考试指导
作　　者：徐孝凯　编著
出 版 者：清华大学出版社(北京清华大学学研大厦，邮编 100084)
http：//www. tup. tsinghua. edu. cn
责任编辑：郑寅堃
印 刷 者：北京市清华园胶印厂
发 行 者：新华书店总店北京发行所
开　　本：787 × 1092 1/16　**印张**：14. 75　**字数**：358 千字
版　　次：2002 年 6 月第 1 版　2002 年 6 月第 1 次印刷
书　　号：ISBN 7-302-05598-X/TP · 3107
印　　数：0001 ~ 6000
定　　价：22. 00 元

丛 书 序 言

高等教育自学考试是我国高等教育重要的组成部分，每年参加自学高考的人数超过百万人次。自学高考中的“计算机及应用专业(专科)”是全国统设专业，是开考时间长、考试人数多的热门专业。该专业目前设置有六门公共基础课，五门专业基础课和五门专业课。在十门专业基础课和专业课中，有“计算机应用技术”、“高级语言程序设计”、“数据库及其应用”、“计算机网络技术”等四门课程与其他相关专业教材共用，另有“模拟电路与数字电路”、“汇编语言程序设计”、“数据结构导论”、“计算机组成原理”、“微型计算机及其接口技术”、“操作系统概论”等六门课程的教材为该专业所专用。本套丛书就是针对上述六门专用课程而组织编写的自学考试指导书。

本套丛书从助学的目的出发，严格按照每门课程自考大纲的要求，帮助和指导每门课程主教材的学习，为此按照以下的特点组织内容：

(1) 除个别章外，每门课程的自学考试指导按章与自考大纲和主教材相对应。

(2) 除个别章外，每章由学习目标、内容提要、重难点辅导、练习题、练习题参考解答等五个部分组成。

(3) “学习目标”部分给出学习本章应达到的目标；“内容提要”部分列出本章所有的知识点，它是对本章内容的高度概括和总结；“重难点辅导”部分对本章较难理解和掌握的内容给予详细透彻地分析和说明；“练习题”部分给出丰富的练习题，通过做练习能够深刻领会和掌握本章所有的知识点；“练习题参考解答”部分给出练习题的全部参考解答。

(4) 附录一给出主教材中每章习题的全部参考解答；附录二给出两套模拟考试试卷及参考解答，以便同学们在复习和考试前演练；附录三给出该课程最近的全国自考考试试卷及参考答案，供同学们借鉴。

针对自学“计算机及应用专业”的广大考生的迫切需要，本着对他们高度负责的精神，我们专门组织了一批既有计算机自考辅导教学经验，又有普通高校计算机教学经验的专家承担编写工作，确保从学习者的角度来组织内容和讨论问题，使本套丛书确实起到助学的作用。

本套丛书专门聘请了一批资深的计算机专家担任审定工作，他们为保证丛书的质量作出了辛勤的努力，在此向他们表示衷心感谢！

欢迎广大读者对本套丛书提出宝贵意见，我们的电子邮件地址为：xuxk@crtvu.edu.cn。

丛书主编　徐孝凯

丛书策划　徐培忠

2002 年 3 月

前　言

数据结构导论课程是全国高等教育自学考试“计算机及应用专业(专科)”一门专业基础课。该课程具有很强的理论性和实践性，是广大自学考生比较难学的一门课程。本人从事多年该课程的教学辅导工作，深知学生学习该课程的指定教材有较大难度，特别是做每章后的习题更难，迫切需要一本可读性好的便于自学的辅导教材，能够引导他们循序渐进地、深入浅出地、有条不紊地、提纲挈领地学习好该课程。本书就是围绕这一宗旨精心编写的，希望通过本书的学习能起到事半功倍的效果。

编写本书的依据：

(1) 全国高等教育自学考试指导委员会制定的“数据结构导论自学考试大纲”。

(2) 全国高等教育自学考试指导委员会组编的《数据结构导论》教材，它是数据结构导论课程的指定用书，由陈小平主编，经济科学出版社出版。

(3) 近年来“数据结构导论”课程的全国自学高考试卷。

本书具有以下特点：

(1) 全书共分八章，按章与该课程自考大纲及指定教材中的内容相对应。

(2) 除个别章外，书中每一章都由学习目标、内容提要、重难点辅导、练习题、练习题参考解答等五个部分组成。

(3) “学习目标”部分给出了学习本章应达到的目标；“内容提要”部分列出了本章所有的知识点，它是对本章内容的高度概括和总结；“重难点辅导”部分对本章较难理解和掌握的内容给予了详细透彻的分析，并通过具体例题加以说明；“练习题”部分根据该课程考试规定的题型给出了丰富的练习题，通过练习能够深刻领会和掌握所要求的知识点；“练习题参考解答”部分给出了上面练习题的全部参考解答，供同学们参考。

(4) 在本书的附录一给出了《数据结构导论》教材中每章习题的全部参考解答，对于较难的习题进行了简要的分析和说明，以便帮助同学们更好地理解和掌握。

(5) 在附录二给出了两套模拟考试试卷及参考解答，以便在复习考试前演练，自测学习成绩和学习效果。

(6) 在附录三给出了该课程最近的全国自考考试试卷及参考答案，供同学们借鉴。

(7) 本书中的所有程序设计算法都在 C/C ++ 语言环境下调试通过，确保了算法的正确性和可靠性。

(8) 本书中的所有程序设计算法都有较好的结构化和模块化设计，同时具有易读性、时间和空间的有效性。通过阅读和学习这些算法，能够养成良好的程序设计风格和习惯。

本人具有较深的数据结构理论和实践知识，主编过多本数据结构教材，为了对自学考生高度负责，本人花费了很大的精力编写这本教材，希望能帮助广大考生学好这门课，取得好的自考成绩。

为了提高本书的质量，专门聘请了具有全国计算机自考教学丰富经验的计算机专家、

北京理工大学教授江涛老师担任主审。江涛教授认真地审阅了全部书稿，提出了很好的意见，特此表示衷心感谢。

由于本人水平有限，不当之处在所难免，敬请广大读者指正，电子邮件地址为：xuxk@crtvu.edu.cn。

本书还可以作为各级各类学校学习“数据结构”课程的教学参考书。

徐孝凯

2002 年 3 月

目　录

第一章　概　　论

一、学习目标

1. 掌握数据、逻辑结构、存储结构、运算、实现等概念的含义。

2. 掌握数据的逻辑结构和存储结构的分类，每一种逻辑结构的特点。

3. 能够根据已知算法或程序段，求出某个语句的执行次数，或求出相应的时间复杂性。

4. 能够用类 C 语言描述处理简单程序设计问题的算法，并求出相应的时间复杂性。

5. 能够区分常量级、线性级、对数级、线性与对数乘积级、平方级、立方级、指数级等算法的时间复杂性的优劣等级。

6. 了解算法和评价的概念，了解算法的空间复杂性的概念。

二、内容提要

1. 利用计算机解决实际问题要实现的两项任务是数据表示和数据处理。

2. 根据解决实际问题的需要，要建立的数学模型包括建立数据的逻辑结构和定义在逻辑结构上的基本运算这两个方面。

3. 数据的存储结构是数据的逻辑结构在计算机存储器上的实现；进行数据处理的算法是在数据的存储结构上进行的对数据的基本运算的实现。

4. 数据由数据元素按照一定的逻辑结构组织而成，而每个数据元素通常又由一个或多个数据项顺序排列而成。同一数据元素的各数据项之间通常无任何次序关系，或者说它们之间为集合结构。

5. 数据的逻辑结构可概括为四种类型：集合结构、线性结构、树形结构和图状结构，由它们的组合可以构成更复杂的逻辑结构。

6. 使数据结构或其中任何元素值发生改变的运算称为“加工”型运算，不影响数据结构或其中任何元素值的运算称为“引用”运算。插入、删除、更新、排序、清空等运算为加工型运算，查找、读取、求元素个数、判空等运算为引用型运算。

7. 数据的存储结构可概括为四种类型：顺序结构、链接结构、索引结构和散列结构，由它们的组合可以构成更复杂的存储结构。

8. 任一种存储结构可以在两个级别(或叫两个层次)上进行讨论，即机器级和语言级，在本教材中采用的是语言级。

9. 实现一个运算的方法和步骤称为此运算的算法。描述一个算法有许多形式，如采用流程图、自然语言、计算机程序设计语言等。在数据结构中主要使用某一种计算机程序设计语言描述。为了简便起见，通常不严格按照计算机程序设计语言的语法规定，只使用其主要成分，而忽略细节，我们把这种算法称为用“伪语言”或“类语言”形式描述的算法。

10. 一个算法的性能主要从四个方面来评价：正确性、可读性、健壮性、有效性(即程序运行时的时间和空间复杂性)。

11. 一个算法的时间或空间复杂性可分为最好、最差(坏)和平均这三种情况讨论，通常只关心最差或平均的情况，并且主要讨论时间复杂性的情况。

12. 算法的时间复杂性的数量级采用大 O 表示，通常有常量级、对数级、线性级、线性与对数乘积级、平方级、立方级、指数级等级别，对应量级表示依次为 $O(1)$，$O(\log_2 n)$，$O(n)$，$O(n\log_2 n)$，$O(n^2)$，$O(n^3)$，$O(2^n)$。当 n 较大时，量级越靠前的算法，其运行时间越短，或者说该算法越有效。

13. 一种数据结构由它的逻辑结构和对它进行的基本运算集组成。数据结构课程研究的主要任务就是各种数据结构的设计和在计算机上的具体实现，即把逻辑结构转化为存储结构，把每一种运算转化为用一种语言描述的算法。

14. 研究数据结构就是研究数据的逻辑结构、存储结构以及相应的运算及其实现。这也是学习数据结构课程的目的与任务。

三、重难点辅导

1. 数据的逻辑结构

数据的逻辑结构一般也叫做数据结构，通常采用二元组描述：

$$B=(D,R)$$

B 表示一种数据结构，它由数据元素的集合 D 和 K 上二元关系的集合 R 所组成。其中

$$D=\{d_i \mid 1\leqslant i\leqslant n,\ n\geqslant 0\}$$

$$R=\{r_j \mid 1\leqslant j\leqslant m,\ m\geqslant 0\}$$

其中 d_i 表示集合 D 中的第 i 个数据元素，n 为 D 中数据元素的个数，特别地，若 $n=0$，则 D 是一个空集，此时 B 也就无结构而言，有时也可以认为它具有任一结构；r_j 表示集合 R 中的第 j 个二元关系(以后均简称关系)，m 为 R 中关系的个数，特别地，若 $m=0$，则 R 是一个空集，表明不考虑集合 D 中的元素之间存在任何关系，彼此是独立的，就像数学中集合里的元素一样。在数据结构的一般讨论中，只考虑 $m=1$ 的情况，即 R 中只包含一个关系($R=\{r\}$)的情况。对于包含有多个关系的数据结构，可分别对每一个关系进行讨论。

D 上的一个关系 r 是序偶的集合。对于 r 中的任一序偶 $<x,y>$ $(x,y\in D)$，我们把 x 叫做序偶的第一元素，把 y 叫做序偶的第二元素，又称序偶的第一元素为第二元素的直接前驱(通常简称前驱)，称第二元素为第一元素的直接后继(通常简称后继)。如在 $<x,y>$ 的序偶中，x 为 y 的前驱，而 y 为 x 的后继。

一种数据结构还能够利用图形形象地表示出来，图形中的每个结点(或叫顶点)对应着一个数据元素，两结点之间带箭头的连线(称做有向边或弧)对应着关系中的一个序偶，其中序偶的第一元素为有向边的起始结点，第二元素为有向边的终止结点，即箭头所指向的结点。

作为例子，下面根据表 1-1 构造出一些典型的数据结构。

表 1-1　教务处人事简表

职工号	姓名	性别	出生日期	职务	部门
01	万明华	男	1952. 03. 20	处长	教务处
02	赵　宁	男	1958. 06. 14	科长	教材科
03	张　利	女	1954. 12. 07	科长	考务科
04	赵书芳	女	1962. 08. 05	主任	办公室
05	刘永年	男	1949. 08. 15	科员	教材科
06	王明理	女	1965. 04. 01	科员	教材科
07	王　敏	女	1962. 06. 28	科员	考务科
08	张　才	男	1957. 03. 17	科员	考务科
09	马立仁	男	1965. 10. 12	科员	考务科
10	邢怀常	男	1966. 07. 05	科员	办公室

表 1-1 中共有 10 条记录，每条记录都由六个数据项所组成，由于每条记录的职工号各不相同，所以可把每条记录的职工号作为该记录的关键字。在下面的例子中，我们将用记录的关键字来代表整个记录。

例 1　一种数据结构 set = (D, R)，其中

D = {01,02,03,04,05,06,07,08,09,10}

R = {}

在数据结构 set 中，只存在有元素的集合，不存在有关系的集合，表明我们只考虑表 1-1 中的每条记录，并不考虑它们之间的任何关系。具有此种特点的数据结构被称为集合结构。在集合结构中，可以把结点之间的关系描述为 0 对 0(0:0)的关系，也就是说，每个结点都是孤立的，它既没有前驱也没有后继。

例 2　一种数据结构 linearity = (D, R)，其中

D = {01,02,03,04,05,06,07,08,09,10}

R = {r}

r = {<05,01>, <01,03>, <03,08>, <08,02>, <02,07>, <07,04>,
<04,06>, <06,09>, <09,10>}

对应的图形如图 1-1 所示。

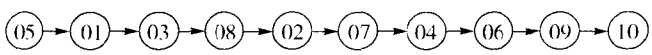


图 1-1　数据的线性结构示意图

结合表 1-1，细心的读者不难看出：r 是按职工年龄从大到小排列的关系。

在 linearity 中，每个数据元素有且仅有一个直接前驱元素(除结构中第一个元素 05 外)，有且仅有一个直接后继元素(除结构中最后一个元素 10 外)。这种数据结构的特点是数据元素之间的 1 对 1(1:1)联系，即线性关系，我们把具有这种特点的数据结构叫做线性结构。

例 3　一种数据结构 tree = (D, R)，其中

D = {01,02,03,04,05,06,07,08,09,10}

R = {r}

r = { <01,02>, <01,03>, <01,04>, <02,05>, <02,06>, <03,07>,
<03,08>, <03,09>, <04,10> }

对应的图形如图 1-2 所示。

结合表 1-1，细心的读者不难看出：r 是人员之间领导与被领导的关系。

图 1-2 像倒着画的一棵树，在这棵树中，最上面的一个没有前驱只有后继的结点叫做树根结点，最下面一层的只有前驱没有后继的结点叫做树叶结点，除树根和树叶之外的结点叫做树枝结点。在一棵树中，每个结点有且只有一个前驱结点(除树根结点外)，但可以有任意多个后继结点(树叶结点可看作为具有 0 个后继结点)。这种数据结构的特点是数据元素之间的 1 对 N(1:N)联系(N≥0)，即层次关系，我们把具有这种特点的数据结构叫做树形结构，简称树。

例 4 一种数据结构 graph = (D, R)，其中

D = {01,02,03,04,05,06,07}

R = {r}

r = { <01,02>, <02,01>, <01,04>, <04,01>, <02,03>,
<03,02>, <02,06>, <06,02>, <02,07>, <07,02>,
<03,07>, <07,03>, <04,06>, <06,04>, <05,07>, <07,05> }

对应的图形如图 1-3 所示。

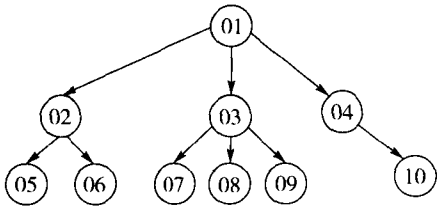


图 1-2 数据的树形结构示意图

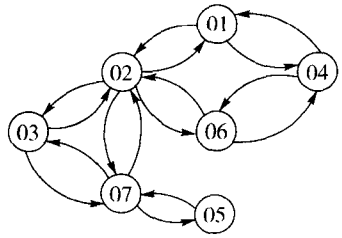


图 1-3 数据的图形结构示意图

从图 1-3 可以看出，r 是 D 上的对称关系，为了简化起见，我们把〈x, y〉和〈y, x〉这两个对称序偶用一个无序对(x,y)或(y,x)来代替；在图形表示中，我们把 x 结点和 y 结点之间两条相反的有向边用一条无向边来代替。这样 r 关系可改写为：

r = {(01,02),(01,04),(02,03),(02,06),(02,07),(03,07),(04,06),(05,07)}

对应的图形如图 1-4 所示。

如果说 r 中每个序偶里的两个元素所代表的人员是好友的话，那么 r 关系就是人员之间的好友关系。

从图 1-3 或图 1-4 可以看出，结点之间的联系是 M 对 N(M : N)联系(M≥0, N≥0)，即网状关系。也就是说，每个结点可以有任意多个前驱结点和任意多个后继结点。我们把具有这种特点的数据结构叫做图形(状)结构，简称图。

从图形结构、树形结构和线性结构的定义可知，树形结构是图形结构的特殊情况(即

M = 1 的情况），线性结构是树形结构的特殊情况（即 N = 1 的情况）。为了区别于线性结构，我们把树形结构和图形结构统称为非线性结构。

例 5 一种数据结构 B = (K, R)，其中

$K = \{k_1, k_2, k_3, k_4, k_5, k_6\}$

$R = \{r_1, r_2\}$

$r_1 = \{<k_3, k_2>, <k_3, k_5>, <k_2, k_1>, <k_5, k_4>, <k_5, k_6>\}$

$r_2 = \{<k_1, k_2>, <k_2, k_3>, <k_3, k_4>, <k_4, k_5>, <k_5, k_6>\}$

若用实线表示关系 r_1，虚线表示关系 r_2，则对应的图形如图 1-5 所示。

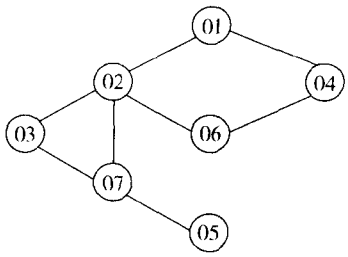


图 1-4 图 1-3 的等价表示

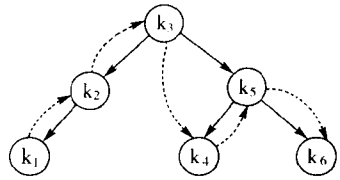


图 1-5 带有两个关系的一种数据结构示意图

从图 1-5 可以看出：数据结构 B 是一种非线性的图形结构。但是，若只考虑关系 r_1 则为树形结构，若只考虑关系 r_2 则为线性结构。

2. 算法描述

算法就是解决特定问题的方法和步骤，它可以借助各种工具描述出来。如从 n 个整数元素中查找出最大值，若用流程图描述则如图 1-6 所示。

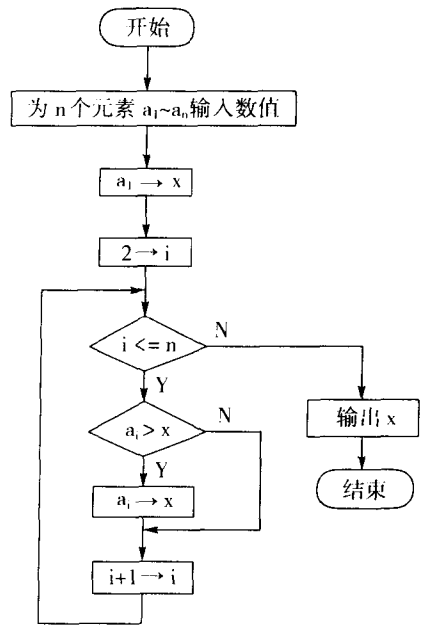


图 1-6 求 n 个元素中的最大值

若采用自然语言（假定使用汉语）描述，则如下列步骤所示：

（1）给 n 个元素 $a_1 \sim a_n$ 输入数值；

（2）把第一个元素 a1 赋给用于保存最大值元素的变量 x；

（3）把表示下标的变量 i 赋初值 2；

（4）如果 $i \leqslant n$ 则向下执行，否则输出最大值 x 后结束算法；

（5）如果 $a_i > x$ 则将 a_i 赋给 x，否则不改变 x 的值，这使得 x 始终保存着当前比较过的所有元素的最大值；

（6）使下标 i 增 1，以指示下一个元素；

（7）转向第（4）步继续执行。

若要使一个算法在计算机上实现，则最终必须采用一种程序设计语言进行描述。如对

于上述算法，采用C语言描述如下：

```
void find()
{
    int i, x, a[n];  //用a[0]~a[n-1]保存a1~an元素
    for(i=0; i<n; i++) scanf(a[i]);
    x=a[0];          //把第一个元素a[0]的值赋给x
    i=1;             //把第二个元素a[1]的下标1赋给i
    while(i<n){
        if(a[i]>x) x=a[i];
        i++;
    }
    printf(x);
}
```

本书采用类C语言进行算法描述，但对C语言有所改进和修改，具体规定如下：

（1）从键盘输入数据通常不采用scanf( )函数调用的形式，而是采用输入标识符cin和提取操作符>>的结合形式。如可将scanf(x, y)语句等价地表示为cin >> x >> y语句，即表示从键盘上依次为变量x和y输入数据。

（2）向显示器屏幕输出信息通常不采用printf( )函数调用的形式，而是采用输出标识符cout和插入操作符<<的结合形式。如可将printf(x, y)语句等价地表示为cout << x << ' ' <<y语句，即表示向屏幕依次输出变量x的值、一个空格和变量y的值。若要向屏幕输出一个换行符，使当前输出位置移到下一行开始，则只要在cout输出语句中使用endl标识符作为输出数据项即可。如cout << x << ' ' <<y << endl语句输出x的值、一个空格和y的值后使输出位置移到下一行开始位置，接着执行的cout语句将从新一行开始显示输出数据项的值。

（3）在函数定义中对参数进行类型说明时，在一个参数的类型标识符和形参名之间使用&符号，则表示该参数为引用参数，不使用该符号的参数为值参数。当调用一个函数时，将把实参的地址传送给对应的引用参数变量，使得引用参数变量成为实参变量的别名，引用参数变量和实参变量访问同一个存储空间，即实参变量所具有的存储空间；把实参表达式的值传送给对应的值参数变量，此时值参数变量具有临时分配的存储空间，在函数体中对值参数变量访问与实参中的变量没有任何关系。不过，若值参为指针变量的话，对指针所指对象的访问实际上就是对实参指针所指对象的访问，因为值参指针与实参指针的值相同(由参数传递时自动赋予的)，它们指向的是同一个对象，即实参指针所指的对象。

（4）通常不使用malloc( )函数得到一个动态存储空间，而是使用new运算符得到一个动态存储空间。如语句int * p = (int *) malloc(sizeof(int)) 的等价表示为int * p = new int，即在new运算符后面写上一个类型标识符就能够得到一个动态存储空间，该存储空间所含字节数等于该类型的长度，同时返回该存储空间的首地址，它自动具有所给类型标识符的指针类型。如执行new double运算时，动态分配具有8个字节的存储空间，返回该存储空间的首地址，它的类型为double *，可以把这个首地址赋给任何一个双精度指针变量。

另外，使用new运算符还能够很方便地分配动态数组空间。如new int[10]语句就分

配了一块能够保存10个整型数的一维数组空间，并返回第一个元素的首地址，它的类型为int *。若把这个首地址赋给int *型指针变量a，则a[0]或*a表示该数组中的第一个元素，a[1]或*(a+1)表示该数组中的第二个元素……a[9]或*(a+9)表示该数组中的第10个元素，即最后一个元素。

(5) 与动态存储分配使用new运算符相对应的是使用delete运算符释放指针所指向的动态存储空间。如要释放p所指向的动态存储空间，可以使用free(p)语句，也可以使用delete p语句。当释放p所指向的动态数组空间时，则使用delete [] p语句。

(6) 对于结构数据类型，被定义后可以直接使用结构名作为结构类型，用来定义该类型的变量，或作为函数的返回类型。如：

```
typedef struct node {
    int data;
    struct node * next;
}Node;
Node x = {5,0};
```

可以采用如下的等价定义：

```
struct Node {
    int data;
    Node * next;
};
Node x = {5,0};
```

可见，后一种定义的结构类型的格式比前一种要简捷得多。上述第一条为结构类型定义语句，定义的结构类型为Node，它包含两个域，一个为整数域data，另一个为Node *域next，它所指向的对象是具有Node类型的对象或者为空值0(NULL)。上述第二条语句定义了一个Node类型的变量x，并对它的data域赋初值5，对它的next域赋初值0，即空值。

(7) 允许在算法的任何地方定义变量，即变量可以随时定义随时使用，当然为了简便起见，变量可以不经过定义而使用，但上机运行时必须添加定义才是正确的。

以上七点规定使算法的描述更加简便，更有助于提高算法的各方面性能，更接近C++语言的描述。因为C++语言正在逐渐取代C语言，现在许多软件开发环境都是面向C++语言的，几乎没有面向C语言的了。

3. 编写算法举例

(1) 交换两个变量值的算法

```
void Swap1(int &a, int &b)
{
    int c = a;
    a = b; b = c;
}
```

执行该函数时，首先把a的值暂存于变量c，接着把b的值赋给a，然后把c的值(即a的原值)赋给b，从而达到交换两个变量a和b值的目的。假定采用如下语句段调用这个

算法：

```
int x=5, y=10;
Swap1(x,y);
cout<<"x="<<x<<' '<<"y="<<y<<endl;
```

在调用前 x 和 y 的值分别为 5 和 10，在调用时分别把实参 x 和 y 对应传送给引用变量 a 和 b，使得 a 是 x 的别名，b 是 y 的别名，在执行函数体时对 a 和 b 的访问实际上就是对相应实参 x 和 y 的访问。函数体执行结束后，x 和 y 的值得到了交换，使得 x 的值为 10，y 的值为 5。执行 cout 语句后得到的打印结果为：

```
x=10  y=5
```

若在上面的函数定义中，不是采用引用参数，而是采用值参数，则必须把 a 和 b 定义为整数指针类型，并在函数体中使用 *a 和 *b 访问 a 和 b 所指对象，这样才能达到交换 a 和 b 所指对象值的目的。此算法如下：

```
void Swap2(int *a, int *b)
{
    int c=*a;
    *a=*b; *b=c;
}
```

当调用该函数时，实参必须是与形参类型相同的值，即整数变量的地址值。若仍然要交换整数变量 x 和 y 的值，则采用的调用表达式为 Swap2(&x, &y)。

在 Swap1 函数定义中，若只是把参数 a 和 b 定义为整型值参，而不是引用，则函数体执行时交换的只是形参 a 和 b 的值，而与调用它的实参 x 和 y 无关，调用前后 x 和 y 的值不会改变。

在函数定义中，当使用结构参数时，通常把它定义为引用参数，因为使用引用参数，调用时不需要为它分配存储空间，进而也省去了把实参的值传送给它的操作和时间，同时还能够在函数体中直接访问实参变量，避免为达到此目的而使用指针传送和访问操作的麻烦。当结构较大(即占用存储字节数较多)时使用引用参数的优点就更加明显。

在函数定义中，若只需要读取引用参数的值，而不需要改变它的值，则可在此参数说明前加上 const 标识符，当在函数体中遇到对它写入数据的操作时，则系统认为非法，将在 C++语言环境下出现编译错误，从而自动保护对应实参的值不会被有意或无意地破坏。如：

```
int Add(const int &x, const int &y)
{
    return x+y;
}
```

在该函数中只能读取 x 和 y 的值，而不允许改变它们的值，否则不能编译和运行。该函数的功能是返回调用它的两个实参变量的值之和。若采用如下语句段调用：

```
int x1=8, x2=12;
cout<<Add(x1,x2)<<endl;
```

则得到的输出结果为20。

（2）建立一个具有 n 个结点的单链表的算法，其中结点类型为上面定义过的 Node 类型。

```
Node * Create(int n)
{
    int i,x;
    Node * p, * q;
    p = new Node;
    q = p;
    for(i = 1;i <= n;i ++ ) {
        cin >> x;
        q -> next = new Node;
        q = q -> next;
        q -> data = x;
    }
    q -> next = NULL;
    return p -> next;
}
```

此算法中的第 1 条语句定义整型变量 i 和 x，其中 i 作为下面的循环变量，x 作为键盘输入变量；第 2 条语句定义两个指向结构类型 Node 的指针变量 p 和 q；第 3 条语句动态分配一个 Node 类型的对象，并将它的首地址赋给 p，或者说使 p 指向这个动态对象；第 4 条语句把 p 的值赋给 q，使 q 也同时指向这个动态对象；第 5 条语句是一个 for 循环，循环体被重复执行 n 次，每次首先从键盘上获取一个整数并送给 x，接着分配一个 Node 类型的动态对象并使 q 的指针域（即 next 域）指向它，再接着使 q 指向这个动态对象，然后把 x 的值赋给这个动态对象的值域（data 域）。此循环执行结束后，可得到由 p 指针所指向的包含有 n + 1 个结点的单链表，其中第一个结点为附加表头结点，它的指针域指向第一个含整数值的结点；第 6 条语句把单链表中最后一个结点的指针域置空；第 7 条语句返回这个单链表的表头指针，它是指向第一个含整数值结点的指针，该单链表含有 n 个结点，每个结点中保存着从键盘上输入的一个相应的整数值。

假定调用该算法时，传给 n 的值为 5，算法运行时从键盘上输入的 5 个整数为 24，13，52，36，48（注：输入的数据之间要用空格分开），则该算法生成的单链表结构如图 1-7 所示，其中 f 表示表头指针。

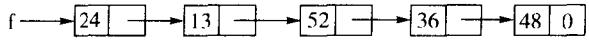


图 1-7　单链表结构示意图

（3）打印输出一个单链表中每个结点的值的算法。

```
void Output(Node * link)
{
    Node * p = link;
    while(p! = NULL) {
        cout << p -> data << ' ';
        p = p -> next;
```