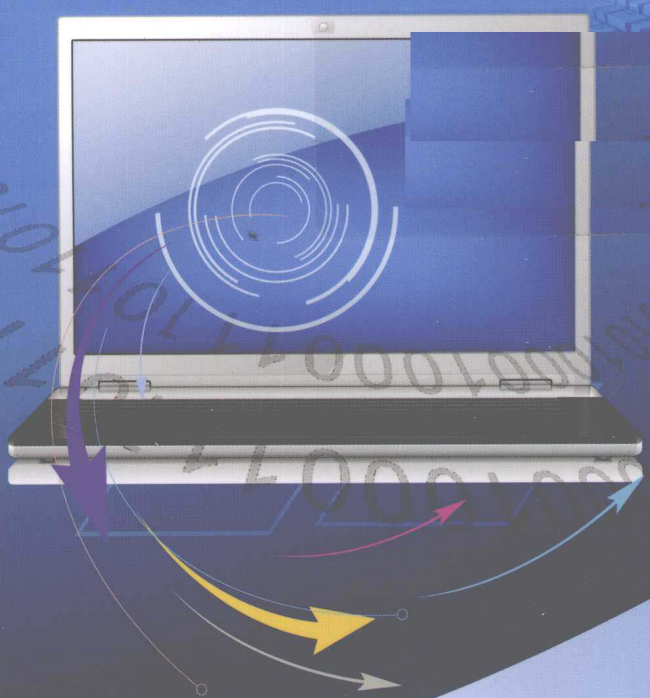
高等学校教材

SHUJU JIEGOU YU SUANFA

数据结构与算法

郑丽英 / 主编



中国铁道出版社
CHINA RAILWAY PUBLISHING HOUSE

高等学校教材

数据结构与算法

郑丽英　主　编
张学军　李金玉　副主编

中国铁道出版社
2010年·北　京

内 容 提 要

本书系统地介绍了数据结构的有关概念、原理、方法和技巧。全书共分10章，以数据抽象类型为主线，首先介绍数据结构的基本概念，然后按照线性表、树、图、查找、排序和文件的顺序，详细介绍了各种数据结构的概念以及所涉及的算法，采用类C/C++语言对相应的存储结构及算法进行了描述，并给出了初步的算法分析。

本书可作为高等院校计算机应用及信息管理专业本科数据结构课程教材，亦可作为相关专业的教学用书，或作为从事软件开发人员的参考书和培训教材。

图书在版编目(CIP)数据

数据结构与算法/郑丽英主编；张学军，李金玉编写. —北京：中国铁道出版社，2010.8

ISBN 978-7-113-11803-7

Ⅰ.①数… Ⅱ.①郑… ②张… ③李… Ⅲ.①数据结构②算法分析 Ⅳ.①TP311.12

中国版本图书馆CIP数据核字(2010)第161223号

书　　名：数据结构与算法

作　　者：郑丽英

责任编辑：武亚雯　李慧君　**电话：**010-51873133　**电子信箱：**tdjc701@126.com　**教材网址：**www.tdjiaocai.com

封面设计：薛小卉

责任校对：孙　玫

责任印制：陆　宁

出版发行：中国铁道出版社（100054，北京市宣武区右安门西街8号）

网　　址：http://www.tdpress.com

印　　刷：北京新魏印刷厂

版　　次：2010年8月第1版　2010年8月第1次印刷

开　　本：787 mm×1 092 mm　1/16　**印张：**16　**字数：**400千

印　　数：0 001～3 000册

书　　号：ISBN 978-7-113-11803-7

定　　价：30.00元

前言

Preface

数据结构是高等院校计算机科学与技术专业的重要核心课程，也是部分非计算机类本科学生和研究生的必修课程。随着计算机软、硬件技术的发展，计算机在各个领域的应用越来越广泛，如数据采集、情报检索、企业管理、搜索引擎、决策分析和人工智能等。而这些领域所面临的首要问题是处理种类繁多、结构复杂的数据及数据关系，因此必须设计高级的数据结构和数据组织，以便有效地采集、组织、存储、传输和处理数据。数据结构主要研究的就是数据的逻辑结构、数据的物理结构以及处理不同结构的数据的算法。

本书以数据的逻辑结构为主线，顺序介绍线性结构、树形结构、图结构和文件结构。在介绍每种数据结构时，进一步讨论其存储结构和相关算法。全书按照抽象数据类型的观点组织，采用 C/C+ + 语言描述，共分 10 章。第 1 章给出了数据结构、抽象数据类型的定义，算法的基本定义及其复杂性表示方法；第 2 章至第 5 章介绍了线性数据结构的内容，给出了线性表、栈、队列、串、数组和广义表的定义、抽象数据类型及其相关的算法；第 6 章和第 7 章内容为非线性数据结构树和图，详细介绍了它们的定义、抽象数据类型，给出了各种物理表示方法及实现；第 8、9、10 章是关于数据处理技术的内容，介绍了几种主要的查找和排序算法，还介绍了文件的概念及其组织方式。

本书的特点是既注重原理又注重实践，对基本概念和原理进行了详细的阐述，对主要数据结构及其算法进行了比较深入的分析讨论，并给出了大量的综合应用。目的在于培养学生合理组织数据和进行优秀的算法设计的能力，使学生能够针对实际问题，利用抽象数据能力，选择合适的数据逻辑结构、存储结构以及相应算法，把数据结构和算法分析以及编程实践相结合起来，在实际中灵活应用。

本书由郑丽英主编，由郑丽英、张学军、李金玉共同编写。其中郑丽英负责第 2 章至第 5 章，张学军负责第 1、6、7 章，李金玉负责第 8、9、10 章。由于作者水平

有限，计算机科学与技术的发展非常迅速，难免存在一些错误和不妥之处，敬请广大读者批评指正。

本书可作为计算机科学与技术专业、软件工程专业的本科生教材，同时适用于其他非计算机专业的数据结构课程的教学；可作为研究生算法分析与设计课程的参考书，同时对计算机科技工作者也有一定的参考价值，特别适于使用 C/C++ 语言编程的科技人员。

本书得到“兰州交通大学计算机科学与技术核心课程”重点课程群项目的资助，在此书稿出版之际，特向兰州交通大学教务处表示感谢。

编　者

2010 年 8 月于兰州交通大学

目录

Contents

1 绪　论

【内容提要】

计算机应用已经深入到人类社会的各行各业，不再局限于早期的科学计算，而更多地应用在如控制、管理等许多非数值数据处理方面。计算机处理的对象也不再是纯粹的数值，而扩展到字符、表格、声音和图像等具有一定结构的数据，这给程序设计带来了一些新的问题，例如，如何组织及存储这些结构性的数据等。为此，必须对程序设计方法进行系统的研究，这不仅涉及研究程序的结构和算法，更重要的是要研究程序的加工对象——数据（信息）的特性以及数据之间的关系（结构）。这就是数据结构这门学科形成和发展的背景。本章讨论数据结构的基本概念，算法及算法分析的基本思想。

1.1 数据结构的基本概念和术语

计算机使用初期，其主要应用领域是科学计算。当我们使用计算机来解决一个具体问题时，一般需要经过下列几个步骤：首先要从该具体问题抽象出一个适当的数学模型，然后设计或选择一个解此数学模型的算法，最后编出程序进行测试、调试，直至得到最终的解答。科学计算问题的数学模型通常可用线性方程组表示，求解的算法相对简单。

然而，随着计算机应用领域的不断扩大，出现了很多无法用数值及数学方法进行描述的问题。这是一类非数值计算问题，下面所举例子就是属于这一类的具体问题。

【例 1.1】 学生成绩查询系统。

该系统的主要功能是根据用户输入的姓名查找相应的成绩。为了完成此功能，该系统必须维持一张成绩表，其一般结构如表 1-1 所示。为了加快查找速度，该成绩表可按姓名首字母有序排列，也可按学号有序排列，表 1-1 为按学号有序排列的成绩表。

表 1-1　学生成绩表

学号	姓名	高等数学	英语	程序设计	平均成绩
9905001	黎明	89	78	90	85
9905002	金菊	70	60	90	73
9905003	何莲	80	70	80	72
9905004	李蓝	70	80	90	80
…	…	…	…	…	…
9905030	李杰	90	80	90	87

我们可以把每一个学生的信息看成是一个记录，表的每个记录又由 6 个数据项组成。该成绩表由 30 个记录组成，记录之间是一种顺序关系。这种表通常称为线性表，数据之间具有线性关系，其主要操作有检索、查找、插入和删除等。

【例 1.2】 八皇后问题。

在八皇后问题中，处理过程不是根据某种确定的计算法则，而是利用试探和回溯的探索技术求解。为了求得合法布局，在计算机中要存储布局的当前状态。从最初的布局开始，一步步地进行试探，每试探一步形成一个新的状态，整个试探过程形成了一棵隐含的状态树，如图1-1所示是一棵简化了的四皇后问题状态树。回溯法求解问题的实质是一个遍历状态树的过程。在这个问题中所出现的“树”也是一种数据结构，它可以应用在许多非数值计算的问题中。

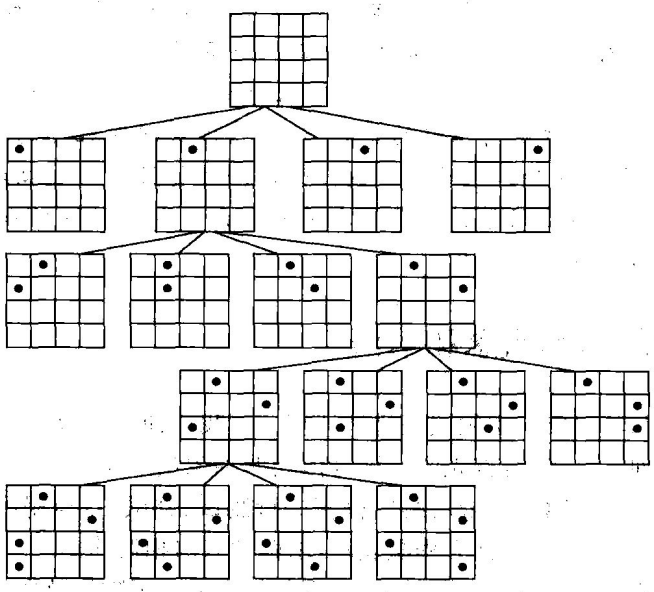


图 1-1 四皇后问题状态树

【例 1.3】 交通咨询问题。

在交通咨询系统中，我们可以采用一种图的结构来表示实际的交通网络。图中的顶点表示城市，边表示城市间的交通联系，对边所赋予的权值可以表示两城市间的距离，或交通费用等等。考虑到交通图的有向性，图中的边可以用弧线来表示，如图 1-2 所示。这个交通咨询系统可以回答旅客提出的各种问题。例如，从某地到某地应如何走才最节省费用，也就是要求从某地到某地的最短路径（当权值为交通费用）。在这个问题中所使用的图也是一种数据结构。

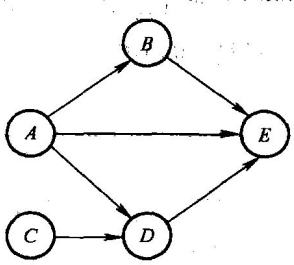


图 1-2 交通网络图

综合上述三个例子可见，描述这类非数值计算问题的数学模型不再是数学方程，而是诸如表、树、图之类的数据结构。因此，可以说数据结构课程主要是研究非数值计算的程序设计问题中所出现的计算机操作对象以及它们之间的关系和操作的学科。

在计算机科学中，数据结构不仅是一般程序设计（特别是非数值计算的程序设计）的基础，而且是设计和实现编译程序、操作系统、数据库系统等系统程序和大型应用程序的重要基础。

在计算机专业中，数据结构是一门综合性的专业基础课程。它不仅是计算机专业教学计划中的核心课程之一，而且是非计算机专业的主要选修课程之一。它可为后续专业基础课提供必要的知识和技能准备，例如程序设计语言及编译技术要使用的栈、散列表以及语法树；操作系统会用到的队列、存储管理表及目录树等；数据库系统运用的线性表、多链表以及索引树等基本数据结构及其一些常用的相关算法。

学习数据结构课程的目的是为了了解能被计算机处理的各种数据对象的特性，了解数据之间的结构特性，而最终目标是在计算机中有效地表示这些具有结构关系的数据，有效地对它们进行处理。

在系统学习数据结构知识之前，我们有必要先介绍数据结构常用的一些基本概念和术语。

1. 数据(Data)

是信息的载体，是描述客观事物的数、字符以及所有能够输入到计算机中并能被计算机识别和处理的符号的集合。数据是计算机程序加工的“原料”。例如，一个代数方程求解程序中所用的数据是整数和实数，而一个编译程序或文本编辑程序中使用的数据是字符串。随着计算机软、硬件技术的发展，应用领域的扩大，数据的含义也随之拓宽。如多媒体技术中所涉及的图形、图像和声音都能被计算机处理，因此，它们也属于数据的范畴。

2. 数据元素(Data Element)

是数据的基本单位，在计算机中通常作为一个整体进行考虑和处理。例如，表 1-1 中每个学生的成绩信息作为数据元素，而图 1-1 中的每一个状态(树中的结点)及图 1-2 中每一个城市信息都可作为一个数据元素。因此，在不同的情况下，数据元素可以是一个字符、一个字符串，也可以是一个由多个数据项组成的记录。有些情况下，数据元素也称为元素、结点、顶点、记录。

3. 数据对象(Data Object)

是性质相同的数据元素的集合，是数据的子集。例如，整数数据对象是集合 $N=\{0,\pm1,\pm2,\pm3,\pm4,\cdots\}$；字母字符数据对象是集合 $C=\{$‘A’, ‘B’, ‘C’, …, ‘Z’, ‘a’, ‘b’, ‘c’, …, ‘z’$\}$。

4. 数据结构(Data Structure)

是指相互之间存在着一种或多种关系的数据元素的集合。它一般包括以下三方面的内容：

(1)数据元素之间的逻辑关系，也称为数据的逻辑结构(Logical Structure)。

(2)数据元素及其在计算机存储器内的表示，也称为数据的存储结构(Storage Structure)。

(3)数据的运算，即对数据施加的操作。

在数据的逻辑结构中，强调数据元素之间的逻辑关系。例如：$a_1, a_2, \ldots, a_{i-1}, a_i, a_{i+1}, \ldots, a_n$ 是一种被称为线性表的数据结构，a_1 是表中的第一个数据元素，a_n 是表中的最后一个数据元素，a_i 是与 a_{i+1} 直接相邻的前一个数据元素，a_i 是与 a_{i-1} 直接相邻的后一个数据元素，这就是线性表数据元素之间的逻辑关系。

数据的逻辑结构是从逻辑关系上描述数据，它与数据的存储无关，是独立于计算机的。因此，数据的逻辑结构可以看作是从具体的问题抽象出来的数学模型。

根据数据元素之间关系的不同特性，通常有下列四类基本的结构(逻辑结构)：

(1)集合结构。在集合结构中，数据元素间的关系是“属于同一个集合”，集合是元素关系

极为松散的一种结构，如图 1-3 所示。

(2)线性结构。该结构的数据元素之间存在着依次排列的一对一的先后次序关系(线性关系)，即除第一个和最后一个数据元素外，其他每个元素有且仅有一个直接前驱和一个直接后继，如图 1-4 所示。

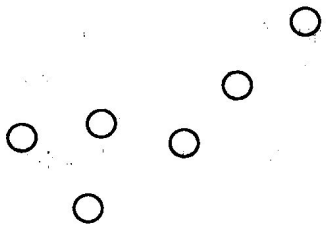


图 1-3 数据的集合关系

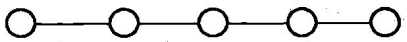


图 1-4 数据的线性关系

(3)树型结构。该结构的数据元素之间存在着一对多的层次关系或分支关系，如图 1-5 所示。

(4)图状结构。该结构的数据元素之间存在着多对多的关系，数据元素之间相互连接成网状，如图 1-6 所示。

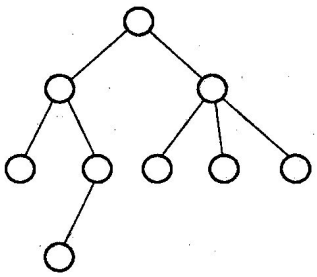


图 1-5 数据的树型关系

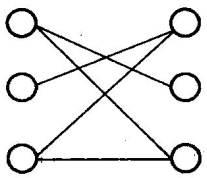


图 1-6 数据的图状关系

数据的运算是定义在数据的逻辑结构上，每种逻辑结构都有一个运算的集合。例如，最常用的运算有：检索、插入、删除、更新、排序等。

数据的存储结构是逻辑结构在计算机存储器里的实现(亦称为映象)，它是依赖于计算机的，对机器语言而言，存储结构是具体的，但我们只在高级语言的层次上来讨论存储结构。存储结构可由以下四种存储方法获得：

(1)顺序存储。该方法是把逻辑上相邻的结点存储在物理位置上相邻的存储单元里，结点间的逻辑关系由存储单元的邻接关系来体现。由此得到的存储表示称为顺序存储结构(Sequential Storage Structure)，通常要借助于程序设计语言的向量来描述。

(2)链接存储。该方法不要求逻辑上相邻的结点在其物理位置上也相邻，结点间的逻辑关系是由附加的指针字段表示的。由此得到的存储表示称为链式存储结构(Linked Storage Structure)，通常要借助于程序设计语言的指针类型来描述。

(3)索引存储。该方法通常是在存储结点的同时，还建立附加的索引表，索引表的每一项称为索引项，索引项的一般形式是：(关键码，地址)，关键码是能够唯一标识一个结点的那些数据项。

(4)散列存储。该方法的基本思想是根据结点的关键码直接计算该结点的地址。

由于存储方式有顺序、链接、索引、散列等多种形式，所以，一种数据结构可以根据应用的需要表示成任意一种或多种存储结构。数据的逻辑结构和存储结构都反映数据的结构，但通常所说的数据结构是指数据的逻辑结构，不包含存储结构的含义。

为了更确切地描述一种数据结构，通常采用二元组表示，即：

$$\text{Data Structure}=(D,R)$$

其中，D 是数据元素的有限集，R 是 D 上关系的集合。

【例 1.4】 一种数据结构 linear_list=(D,R)，其中

$D=\{1,2,3,4,5,6,7\}$

$R=\{r\}$

$r=\{<1,2>,<2,7>,<7,3>,<3,5>,<5,4>,<4,6>\}$

对应的图形如图 1-7 所示。

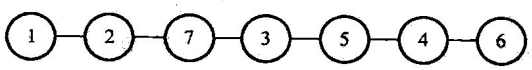


图 1-7 数据的线性结构示意图

显然，这里的 r 表示的是一种线性关系。

【例 1.5】 一种数据结构 tree=(D,R)，其中

$D=\{1,2,3,4,5,6\}$

$R=\{r\}$

$r=\{<1,2>,<1,3>,<1,4>,<2,5>,<2,6>\}$

对应的图形如图 1-8 所示。这里的 r 表示的是一种一对多的层次关系。

【例 1.6】 一种数据结构 graph=(D,R)，其中

$D=\{1,2,3,4,5,6\}$

$R=\{r\}$

$r=\{(1,6),(2,6),(6,5),(6,4),(4,5)\}$

对应的图形如图 1-9 所示。这里的 r 表示的是一种多对多的网络关系。

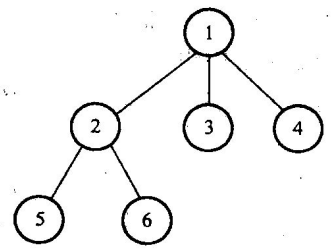


图 1-8 数据的树结构示意图

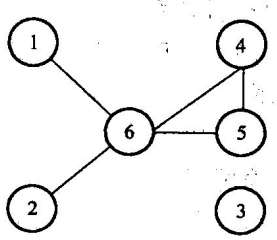


图 1-9 数据的图结构示意图

【例 1.7】 一种数据结构 $N=(D,R)$，其中

$D=\{a,b,c,d,e,f\}$

$R=\{r1,r2\}$

$r1=\{<c,b>,<c,e>,<b,a>,<e,d>,<e,f>\}$

$r2=\{<a,b>,<b,c>,<c,d>,<d,e>,<e,f>\}$

若用实线表示关系 $r1$，虚线表示关系 $r2$，则对应的图形如图 1-10 所示。

这里，N 是一种非线性的图型结构。但若只考虑关系 $r1$ 则为树型结构，若只考虑关系 $r2$ 则为线性结构。

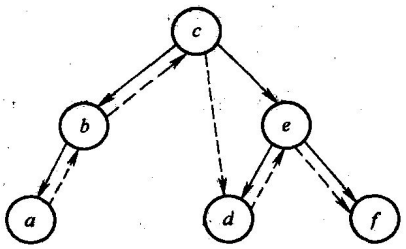


图 1-10 带有两个关系的一种数据结构示意图

5. 数据类型(Data Type)

数据类型是和数据结构密切相关的一个概念，它最早出现在高级语言中，在高级语言的程序设计中，每个常量、变量或表达式都有确定的数据类型。数据类型明显或隐含地规定了在程序执行期间，变量或表达式所有可能的取值范围，以及在这些值上允许进行的操作。因此数据类型是一个值的集合和定义在这个值集合上的一组操作的总称。例如整数类型(int)，它的值集范围是[−32 768,32 767]，定义在其上的一组操作为加、减、乘、除及取模等。

在高级语言中，数据类型可分为两类：一类是原子类型，另一类是结构类型。原子类型的值是不可分解的。例如，C 语言中的整型、实型、字符型都是原子类型。而结构类型是由若干成分按某种结构组成的，因此是可分解的，并且它的成分可以是非结构的，也可以是结构的。例如数组由若干分量组成，每个分量可以是整数，也可以是数组等。在某种意义上，数据结构可以看成是“一组具有相同结构的值”，而数据类型则可看成是由一种数据结构和定义在其上的一组操作组成的。

6. 抽象数据类型(Abstract Data Type，简称 ADT)

抽象数据类型是指数学模型以及定义在该模型上的一组操作。抽象数据类型和数据类型实质上是一个概念。例如，各种计算机都拥有的整数类型就是一个抽象数据类型，尽管它们在不同的处理器上的实现方法不同，但由于其定义的数学特性相同，在用户看来都是相同的。因此，“抽象”的意义在于数据类型的数学特性。但在另一方面，抽象数据类型的范畴更广，它不再局限于现有程序设计语言中已实现的数据类型(通常称为固有数据类型)，还包括用户在设计软件系统时自己定义的数据类型。为了提高软件的复用率，在近代程序设计方法学中，要求在构成软件系统的每个相对独立的模块上，定义一组数据和施与这些数据的一组操作，并在模块的内部给出这些数据的表示及其操作的细节，而在模块的外部使用的只是抽象的数据及抽象的操作。因此，抽象数据类型的最重要的特点是抽象和信息隐蔽。抽象的本质就是抽取反映问题本质的东西，忽略非本质的细节，从而使所设计的数据结构更具一般性，可以解决一类问题。信息隐蔽就是对用户隐藏数据存储和操作实现的细节，使用者仅需了解抽象操作，或界面服务，通过界面中的服务来访问这些数据。

抽象数据类型的定义可以由一种数据结构和定义在其上的一组操作组成，而数据结构又包括数据元素及元素间的关系，因此抽象数据类型一般可以由元素、关系及操作三种要素来定义。和数据结构的形式定义一样，抽象数据类型可用以下的三元组表示：

ADT=<D,S,P>

其中，D 表示数据对象，S 是 D 上的关系，P 是 D 的基本操作集。

本书采用以下格式定义抽象数据类型。

ADT <抽象数据类型名> is

数据对象：<数据对象定义>

数据关系：<数据关系定义>

基本操作：<基本操作的定义>

End <抽象数据类型名>

例如，栈的 ADT 定义为：

ADT Stack is

数据对象：属于同一个数据对象集合。

数据关系：具有线性关系。

基本操作：

(1)InitStack(&S)：创建一个空栈 S；

(2)StackEmpty(S)：判断栈 S 是否为空，若是空则返回真，否则返回假；

(3)ClearStack(&S)：清空栈 S 中的所有元素，使之成为一个空栈；

(4)GetTop(S)：返回 S 的栈顶元素；

(5)Push(&S,e)：元素 e 进栈，成为 S 的栈顶元素；

(6)Pop(&S)：删除 S 的栈顶元素；

(7)DestroyStack(&S)：销毁栈 S。

} ADT Stack

抽象数据类型的特征是使用与实现分离，实现封装和信息隐藏。即在抽象数据类型设计时，把类型的声明与其实现分离开来。

抽象数据类型的实现依赖于所选择的高级语言的功能。从程序设计的历史发展来看，有几种不同的实现方法。一种是传统的面向过程的程序设计，也就是我们现在所常用的方法，根据数据的逻辑结构选定合适的存储结构，根据所要求的操作设计出相应的子程序或子函数。另一种是“包”、“模块”的设计方法，Ada 语言提供了“包”、“模块”(Module)结构，TURBO PASCAL 语言提供了“单元”(Unit)结构，用这类结构实现 ADT 比起第一种方法有一定的进步。第三种是面向对象的程序设计(Object Oriented Programming，简称 OOP)，在面向对象的程序设计语言中，存储结构的说明和操作函数的说明被封装在一个整体结构中，这个整体结构称为“类”(Class)，属于某个“类”的具体变量称为“对象”(Object)。理论上说，用面向对象的程序设计语言，如 C++描述抽象数据类型更合理，更直观，因为面向对象中的类就是抽象数据类型。

1.2 算法及算法分析

1.2.1 什么是算法

算法(Algorithm)是对问题求解过程的一种描述，是为解决一个或一类问题给出的一个确定的、有限长的操作序列。

一个算法应该具有下列特性：

(1)有穷性，一个算法必须在执行有穷步之后结束。

(2)确定性，算法中的每一步都必须具有确切的含义，无二义性。

(3)可行性，算法中的每一步都必须是可行的，也就是说，每一步都能通过已经实现的基本运算的有限次执行得以实现。

(4)输入，一个算法可以有一个或多个输入，这些输入取自特定的数据对象集合。

(5)输出，一个算法具有一个或多个输出，这些输出与输入之间存在某些特定关系。

在计算机领域，一个算法实质上是针对所处理问题的需要，在数据的逻辑结构和存储结构的基础上施加的一种运算。由于数据的逻辑结构和存储结构不是唯一的，在很大程度上可以由用户自行选择和设计，所以处理同一个问题的算法也不是唯一的。另外，即使对于具有相同的逻辑结构和存储结构而言，其算法的设计、思想和技巧不同，编写出的算法也大不相同。我们学习数据结构这门课程的目的，就是要学会根据处理问题的需要，为待处理的数据选择合适的逻辑结构和存储结构，进而按照结构化、模块化以及面向对象程序设计方法设计出比较满意的算法(程序)。

要设计一个好的算法通常要考虑以下要求：

(1)正确性，选择或设计的算法首先应该是正确的。正确性是设计和评价一个算法的首要条件。如果设计的算法不正确，其他方面就无从谈起。一个正确的算法是指在合理的数据输入下，能在有限的运行时间内得出正确的结果。

(2)可读性，一个算法应该便于阅读和交流。可读性好不仅有助于人们对算法的理解，而且也有助于程序的调试与修改。在保证算法正确的前提下，应强调算法的可读性。

(3)健壮性，健壮性是指一个算法对不合理输入(不正确、非法、错误)的反映和处理能力。一个好的算法应该能够识别出错误数据并进行相应处理，而不会发生异常或莫名其妙的结果。

(4)高效性，是指算法的执行效率要高。算法的效率包括时间效率和空间效率两个方面。时间效率是指算法的执行所需要的时间，而空间效率是指执行该算法所需要的存储量。对于同一个问题和同一问题规模，若采用不同的算法进行处理，其执行的效率就可能不相同。对同一种算法来说，如果执行的时间越短，所需的存储越小，则其效率越高。

1.2.2 算法的描述

算法可用自然语言、框图或高级语言进行描述。用自然语言描述算法简单且便于被人理解，但易于产生二义性。框图直观但不擅长表达数据的组织结构，而高级程序语言描述算法则较为准确且严谨。高级程序语言描述的算法可直接在计算机上运行，从而使给定问题在有限时间内被机械地求解。不过直接使用高级语言来描述算法，也有语言细节过多、直观性差，常常需要借助于注释才能使人看明白的弱点。为此，常常采用伪码语言(也称类程序设计语言)描述算法。伪码语言介于程序语言和自然语言之间，它忽略程序语言中一些严格的语法规则与描述细节，以便把注意力主要集中在算法处理步骤的描述上。因此，它比程序语言更容易描述和被人理解，而比自然语言更接近程序语言。伪码语言算法不能直接在计算机上执行，但容易编写和阅读，并且很容易转换为可在计算机上运行的程序。例如，在很多数据结构教材中使用的类 PASCAL 或类 C 语言都是伪码语言。

本教材采用类 C/C++语言进行算法描述，类 C 是对 C 语言的一种化简，不但保留了 C 语言的精华而且与标准 C 语言兼容。此外，引入了部分 C++语言的表示方法，使算法的表达更简洁直观，容易理解。所有的算法都很容易转换为可执行的 C 语言或 C++语言程序。以下仅对部分语法成分做简要说明。

(1)注释，采用 C++语言的注释方法，其格式如下：

```
//  注释内容
```

(2)输入和输出，采用 C++语言的流式输入输出的形式，其相关格式如下：

```
cin ≫变量 1≫变量 2≫…≫变量 n;
```

表示输入一个数据，存放到右部变量里。

cout ≪变量 1≪变量 2≪…≪变量 n;

表示输出右部变量的值。

(3)数据结构(存储结构)的表示用类型定义(typedef)描述。数据结构中的很多类型是由若干相同或不同类型的常量或变量,按照一定的方式组合而形成。一般使用结构体来描述这种组合类型,其格式如下:

```
typedef struct node{
  int data;
  struct node * next;
  }Node;
```

用 Node list_node 来说明变量 list_node 的类型是 struct node 类型,它包括数据域 data 和指针域 next,指针域是指向结构体类型的指针变量。本书均采用这种方式来描述组合数据类型。

(4)数据元素类型约定为 ElemType,由用户在使用该数据类型时自行定义,其格式如下:

typedef 数据类型 ElemType;

数据类型可为基本数据类型,如 int,也可为组合数据类型。

(5)基本操作的算法都用以下形式的函数描述:

```
函数返回类型 函数名(函数参数表){
  //  算法说明
   语句序列;
} //函数名
```

为了便于算法描述,除了值调用方式外,增加了 C++语言的引用调用的参数传递方式,引用参数可被函数本身修改参数值,可以此作为输出数据的管道。

在 C++中可以通知编译器对函数的一个或多个参数产生自动的引用调用而不是值调用。实现方法就是在函数说明的参数表中,在参数名前面加一个引用运算符 &。例如,函数f( )的参数 x 是一个引用类型,类型说明为 ElemType &x,ElemType 定义为 int。函数 f( )的说明如下:

```
void f(ElemType &x)
{
  x+ = 50;
}
```

这种函数的说明也可用于函数原型中。定义引用参数后,当发生函数调用时,C++编译器就自动地使实参与形参共用同一个地址。例如:

```
void main( )
{
  int x1= 50;
  f(x1);
  cout≪x1;
}
```

在上述函数调用语句执行时,函数 f( )的参数 x 就是其相应的实参 x1 的引用,即形参 x

是其对应实参 x1 的别名，它们共用一个地址。因此，函数 f( )对形参 x 的操作就是对实参 x1 的操作。上述程序执行后，输出 x1 的值是 100，而不是 50。

(6)内存的动态分配与释放

C 语言是用函数 malloc( )和 free( )动态分配内存和释放内存。本书采用 C++中使用的操作符 new 和 delete 动态分配和释放内存空间，其一般形式如下：

指针变量名＝new 数据类型名；

delete 指针变量名；

new 的功能是分配存放“类型名”所指定类型的内存并返回指向该内存的地址；delete 的功能是释放由指针变量所指向的内存。

(7)其他

由于 C 语言没有定义布尔类型，本书的算法描述中，经常用到布尔值，因此这里定义一个枚举类型：

```
enum boolean{FALSE,TRUE};
```

此外，还有其他的一些语法，这里就不一一给出了，相信熟悉 C 语言的读者能够自行处理。

1.2.3 算法分析

求解同一个问题，可以有许多不同的算法，根据什么标准认为哪一个更好呢？通常考虑以下三个方面：

(1)耗费时间短。

(2)占用空间少。

(3)易于理解，易于编码，易于调试，易于推广。

在实际应用当中，很难做到面面俱到。要节省时间往往就要牺牲空间；要舍不得空间就要牺牲时间。因此我们只能根据具体情况有所侧重。若该程序使用次数较少，则力求算法简明易懂；对于反复多次使用的程序，应尽可能选用快速的算法；若待解决的问题数据量极大，机器的存储空间较小，则相应算法主要考虑如何节省空间。

评价算法的准则涉及一些因素，例如简洁性、易读性以及是否适应所处理的问题等，其中比较客观的一个因素是运行时间。确定算法的运行时间的一个可行的办法是做实验：将算法编成程序，并在某个特殊的计算机上运行某些选定的数据，测量运行时间。但是，这种方法的缺点是运行时间不仅与算法有关，而且与计算机的指令集、编译系统的质量以及所选择的数据等有关。为了独立于机器的软、硬件系统来分析算法的时间耗费，可利用语句的“频度”和算法的渐进时间复杂度来讨论算法执行的时间消耗。下面首先考虑两个 n 阶方阵相乘的例子：

【例 1.8】 求两个 n 阶方阵的乘积 $C=AB$。

```
void MATRIXMLT(float A[n][n],float B[n][n], float C[n][n])
{ //求两个 n 阶方阵的乘积 C=AB
 int i,j,k;
(1) for(i= 0;i< n;+ + i)                    n+ 1
(2) for(j= 0;j< n;+ + j)                    n(n+ 1)
(3) { C[i][j]= 0;                           n²
```