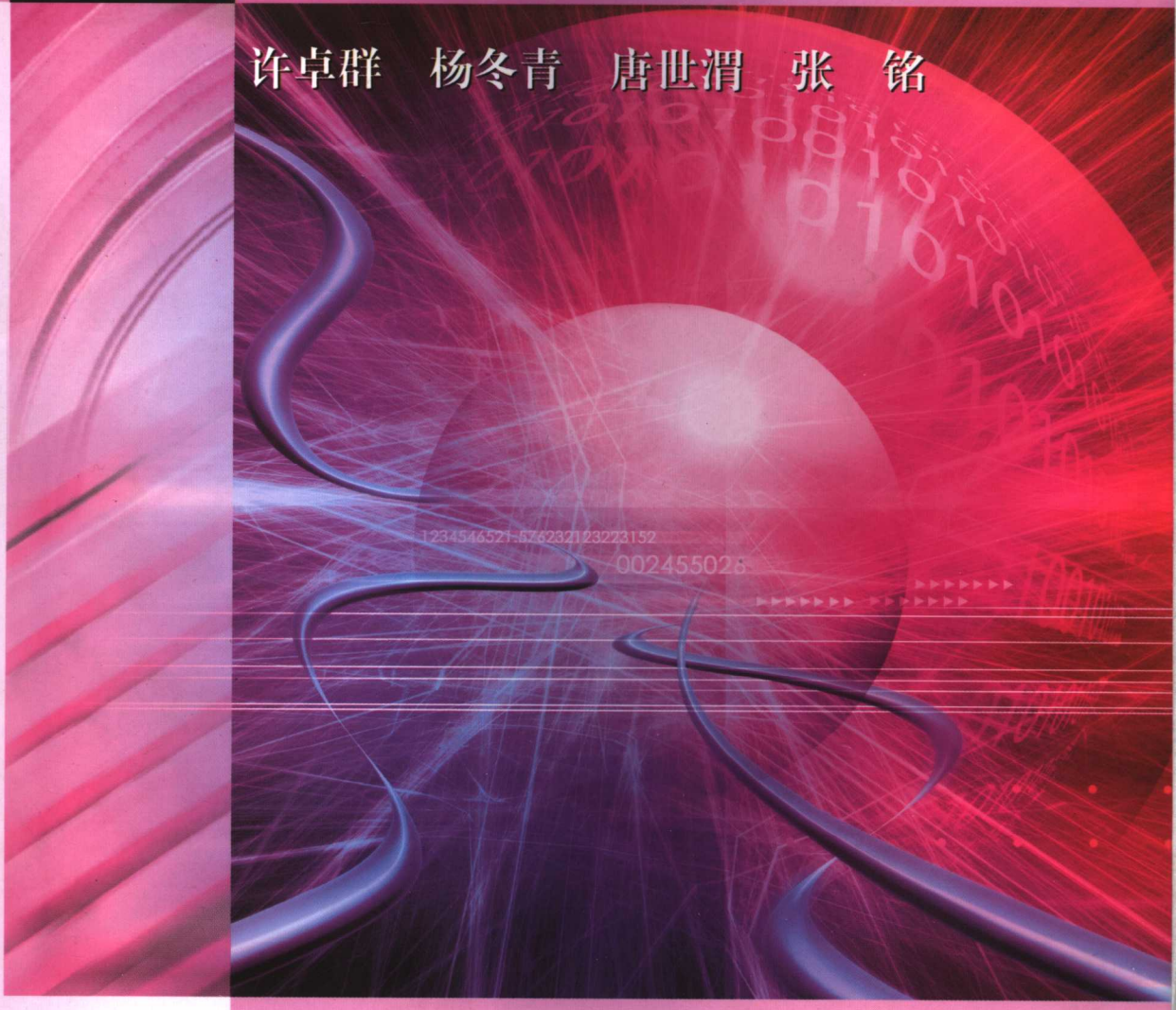
普通高等教育“十五”国家级规划教材

数据结构与算法

许卓群　杨冬青　唐世渭　张　铭



高等教育出版社

普通高等教育“十五”国家级规划教材

数据结构与算法

许卓群　杨冬青　唐世渭　张　铭

高等教育出版社

内容提要

本书把数据结构的原理和算法分析技术有机地结合在一起，系统地介绍了各种类型的数据结构和排序、检索的各种算法，还引入了一些比较高级的数据结构及相关的算法分析技术。

本书分为基本数据结构、排序和检索、高级数据结构三部分。借助抽象数据类型，从逻辑结构的角度系统地介绍了线性表、字符串、二叉树、树和图等各种基本数据结构；从算法的角度讨论排序、检索和索引算法；从应用的角度介绍了一些复杂的线性表结构、复杂树结构以及空间数据结构。本书采用能够自然体现抽象数据类型概念的C++语言作为算法描述语言，注意对每一种数据结构的不同存储方法与有关算法进行比较分析。很多算法使用了参数化的模板，从而提高算法中数据类型的通用性，支持高效的代码重用。

本书注意对概念的清晰引入，论述上加强逻辑性，并增加了一些新颖内容。本书可作为高等院校计算机及相关专业学生的教材和参考书，也可供从事计算机的工程技术人员学习参考。

图书在版编目（CIP）数据

数据结构与算法 / 许卓群等. —北京：高等教育出版社，2004.7(2006 重印)
ISBN 7-04-014616-9

Ⅰ.数... Ⅱ.许... Ⅲ.①数据结构②算法分析
Ⅳ.TP311.12

中国版本图书馆 CIP 数据核字（2004）第 059788 号

策划编辑 刘建元 **责任编辑** 武林晓 **市场策划** 陈 振
封面设计 于文燕 **责任印制** 杨 明

出版发行	高等教育出版社	**购书热线**	010-58581118
社 址	北京市西城区德外大街 4 号	**免费咨询**	800-810-0598
邮政编码	100011	**网 址**	http://www.hep.edu.cn
总 机	010-58581000		http://www.hep.com.cn
		网上订购	http://www.landraco.com
经 销	蓝色畅想图书发行有限公司		http://www.landraco.com.cn
印 刷	国防工业出版社印刷厂	**畅想教育**	http://www.widedu.com
开 本	787×1092 1/16	**版 次**	2004 年 7 月第 1 版
印 张	30	**印 次**	2006 年 2 月第 3 次印刷
字 数	620 000	**定 价**	29.50 元

物料号 14616-00

前　言

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。例如，对于线性表，讨论数组和链表等相关的存储结构；结合应用特点和线性表运算特点，讨论了向量、栈和队列。对于一些重要算法，列出了单独章节来讨论，例如排序、检索、索引等。

数据结构这门课程的三个教学目标是：第一，让学生懂得“数据结构＋算法＝程序”，也就是说，程序不仅仅是算法的实现。求解问题要恰当设计数据结构，并把它和求解算法结合起来。第二，培养学生数据抽象的能力，不仅让学生了解链表、树、图等数据结构是如何实现的，特别要加强对数据逻辑关系的分析与认识。第三，要把数据结构与算法的理论分析与编程实践相结合，在实际应用中灵活应用。综合性的上机项目对于达到第三个目标是很有帮助的。

总之，数据结构和算法需要强调以下四个方面的知识和能力：(1) 掌握并能够灵活应用基本数据结构的抽象数据类型、存储方法和主要的算法，特别是线性结构、二叉树、树、图、文件等；(2) 掌握并应用常用的排序、检索和索引算法和方法；(3) 掌握基本的算法设计和分析技术，并结合具体的设计，对所设计的数据结构和算法进行分析；(4) 在进行程序设计、调试、测试的项目训练(即上机实习训练)中，注意综合应用，将所学到的数据结构和算法知识应用到对具体数据对象的特性分析中。结合实际例子进行设计，选择合适的数据结构和存储结构以及相应的算法。显然，合理地组织数据、有效地表示数据、有效地处理数据，这三者是提高程序设计质量的关键因素，它们为提高程序的清晰性和易读性等创造了基础。

本书的特点是注意基本概念的引入和阐述，对主要数据结构及其相关算法分析讨论比较深入。多年来，这些内容一直作为北京大学计算机科学技术系的数据结构课程(本科专业主干基础课程)的主要教学内容。本书是多年教学经验的总结，与本书直接有关的教材(1987 年由高等教育出版社出版的《数据结构》)曾于 1992 年获得教育部优秀教材一等奖。因此，作者在编写本书时很注意继承《数据结构》的优点，同时又大范围地进行了内容更新。数据结构是一门理论和实践结合比较紧密的课程，本书每一章中都设计了比较新颖的综合上机实习题(即项目实习)，因此可以说，这是一本具有崭新内容的教学用书。它可以作为高等学校计算机类专业学生的教材和参考书，也可作为其他理工类专业的数据结构课程的教学用书。

本书的第 1、2、3 章由许卓群教授执笔；第 4、5、6 章由杨冬青教授执笔；第 8 章和第 10 章由唐世渭教授执笔；第 7、9、11、12 章由张铭副教授执笔。本书的很多内容曾经由张铭副教

授在2002和2003学年度给北京大学计算机科学技术系讲授的数据结构课程中使用，并对各章提出了详细的修改意见。听课的本科生以及参与课程辅导的01级、02级硕士研究生们对书稿和算法代码提出了许多宝贵的修改意见，在此一并表示感谢。

尽管本书经过了作者反复讨论和推敲，并经历了教学实践的检验，但限于水平，难免有不妥之处，希望读者不吝赐教。为了读者联系方便，作者将开放北京大学信息科学技术学院数据结构网站 http://db.pku.edu.cn/mzhang/ds/index.htm。网站主要内容有：多媒体演示讲稿、标准C++模板编写的可执行的源程序代码、习题和上机题及其参考答案。课程网站还有一个BBS讨论版，读者可以在BBS上提出问题、建议和意见，作者将组织助教回答问题、讲解习题中的要点和难点等。

许卓群 杨冬青 唐世渭 张 铭

2004年5月于北京大学

目　录

第1章 概 论

1.1 为什么要学习数据结构

“数据结构与算法”是计算机科学与技术学科的一门专业基础课程，它可为后续的专业课程学习提供必要的知识和技能准备。计算机科学与技术的很多后续课程都要用到本书所涉及的知识和技能，例如，程序设计语言及其编译技术要使用栈、散列表及语法树，操作系统会用到队列、存储管理表及目录树，数据库系统将运用线性表、多链表以及索引树等基本数据结构及其相关的算法。本课程讨论的其他一些数据结构，如广义表、集合以及图的搜索等也是很多应用领域经常涉及的。本课程的目的是介绍一些最常用的数据结构，阐明数据结构内在的逻辑关系，讨论它们在计算机中的存储表示，并结合各种数据结构，讨论对它们实行各种运算（操作）的实现算法。很多算法实际上是对某种数据结构施行的一种变换，研究算法也就是研究在施行变换过程中数据结构的动态性质。以上这些知识再配合以实际的上机编程练习，将增强读者求解复杂问题的能力，其中包括根据问题的性质恰当选择数据结构的能力，以及控制求解算法的复杂性的能力。对问题求解复杂性的估计将涉及求解问题的空间复杂性（对算法所使用的存储空间资源需求的估计）和求解问题的时间复杂性（对根据算法所施行的计算过程的计算时间估计，亦即占用 CPU 资源的估计）。

本章将讨论在后续各章都用到的预备知识和一些设计原则。限于内容安排，一些内容讨论还不够深入，后续章节还会继续讨论，请参照阅读。

1.2 什么是数据结构

数据结构（data structure）涉及数据之间的逻辑关系、数据在计算机中的存储方式和在这种结构上的一组能行的操作（运算）三个方面。为了叙述方便，把上述三个方面分别称为：数据的逻辑结构、数据的存储结构和数据的运算。

1.2.1 数据的逻辑结构

数据的逻辑结构（logical structure）反映了人们对数据的含义解释。人们对现实世界的某个特定领域的认识，表现在对所涉及的事物之间的逻辑关系以及组成结构的认识，这些认识

可以部分地反映为数字世界中数据间的逻辑关系和逻辑结构。用集合论的观点来看，数据的逻辑结构由数据结点(node)和连接两两结点的连接边(edge)两类元素组成。一个逻辑结构可以用一组数据结点(表示为结点集合 K)和一个关系集合 R 来表示：

$$(K, R)$$

其中，K 是由有限个结点组成的集合，R 是一组定义在集合 K 上的二元关系 r，其中的每个关系 $r(r \in R)$都是 $K \times K$ 上的二元关系(binary relation)，用来描述结点(数据)之间的逻辑关系。举例来说，讨论某家族的成员，把每个成员个体的描述作为数据结点，而全部成员组成结点集 K。家族的各类亲属关系就是 R，如母系血缘关系 r、远亲关系 r^* 和非血缘的亲情关系 r' 等等，每一个关系可以用成员的关系元组(连接边)来给出，例如，母子关系(王爱莲，张选)等等。再以学生的周课程表为例，将每天的课程安排数据作为结点，一共引入 7 个结点，它们的名称依次为“星期一”，“星期二”，“星期三”，…，“星期日”。按照常规含义，这些结点两两之间显然有一个时间先后关系 r，$r = \{$(“星期一”，“星期二”)，(“星期二”，“星期三”)，(“星期三”，“星期四”)，(“星期四”，“星期五”)，(“星期五”，“星期六”)，(“星期六”，“星期日”)$\}$。此集合中的 6 个元组描述了“时间先后”关系 r。此外，为了表示星期日和下一周工作日的时间顺序，还可以引入另一个关系 $r^* = \{$(“星期日”，“星期一”)$\}$，r^* 只含有一个元组实例(“星期日”，“星期一”)，其含义是说，星期日之后是另一个星期一。

在上述的实例中，对结点本身的内容没有什么限制。每个结点可以是很复杂的数据类型。例如，可以为以上 7 个结点的结点数据类型引入属性 weekday，让其中的 5 个结点取属性值 weekday = true，而另两个结点取值为 weekday = false。此外，为了记录每天的课程，可以在结点数据中存储当日课程的名称，时间和地点等。

1. 结点的类型

结点的数据类型可以是基本数据类型，也可以是复合数据类型。常见的程序设计语言中使用了 5 种基本的数据类型，它们的共同特点是：程序在访问数据时，把基本数据类型看做一个整体，而不会把基本类型的一个组成部分看做独立含义的数据。

- 整数类型(integer)：该类型规定了所能表示的整数范围，由于在计算机中一般使用 1 到 4 个字节来存储整数，所以整数类型所能表示的整数范围也严格限制在它的存储字节所能表示的整数范围内。
- 实数类型(real)：计算机的浮点数据类型所能表示的数值范围和精度是有限的，一般使用 4 到 8 个字节来存储浮点数，所以实数类型所能表示的范围和精度也严格限制在该类型的存储字节所能表示的范围内。
- 布尔类型(boolean)：取值为真(true)或假(false)，在 C++ 语言中，一般使用整数 0 表示假，用非 0 表示真。
- 字符类型(char)：用单个字节(8bit，最高位 bit 为 0)表示 ASCII 字符集中的字符。字符类型不包括汉字符号。为了简化讨论，本书在涉及字符类型时一般限于这种单字节的字符

类型。但是需要说明的是,在涉及中文处理时,由于汉字符号需要使用2个字节(每个字节的最高位bit为1)的编码,单个字节对于汉字是没有独立含义的,因此在中英文混合的文字处理中,单字节的ASCII符号和双字节的汉字混合排列,如果不注意就会引起乱码。在C++中把双字节表示中文符号的字节类型称为w_char类型(wide character)。目前,国际上已经采用了统一的扩展字符集合标准UNICODE,这一标准允许英、日、韩、阿拉伯语等文字的混合文字处理。由于涉及多字节符号编码的细节,本书就不再进一步讨论它。

• 指针类型(pointer):该类型用于表示机器内存地址,即表示指向某一内存单元的地址。由于机器的指令系统一般采用32 bit或64bit的地址长度,所以指针类型也相应地用4或8个字节来表示一个指针。指针值的存储和指针值的运算方式在形式上与正整数相似。但指针的运算一般仅限于两个指针地址的比较、加减或对一个指针增减一个整数量等。

复合类型是由基本数据类型组合而成的数据结构类型。例如,在程序语言中的数组类型、结构(记录)类型等皆属于复合数据类型。复合数据类型本身又可以参与定义结构更为复杂的结点类型。总之,结点的类型不限于基本数据类型,可以根据应用的需要来灵活定义。

2. 结构的分类

讨论逻辑结构(K,R)的结构分类,一般以关系集R的分类为主。假定关系R集合仅含一个关系r,用r的性质来刻画数据结构的特点,进行分类。

• **线性结构(linear structure)**:这种结构在程序设计中应用最多。它的关系r是一种线性关系,或称为"前后关系",有时也称为"大小关系"。关系r是有向的,且满足全序性和单索性等约束条件。全序性是指,线性结构的全部结点两两皆可以比较前后(关系r)。单索性是指,每一个结点x都存在惟一的一个直接后继结点y。如果其他结点z在y之前,则这个z也一定在x之前,不会在x,y之间。

• **树形结构(tree structure)**:简称树结构,或称为层次结构,其关系r称为层次关系,或称"父子关系"、"上下级关系"等。这一关系r要比上述的线性关系的约束少一些。每一个结点可以有多于一个的"直接下级",但是它只能有惟一的"直接上级"。可以把树形结构类比为家族的家谱层次结构,每个结点只允许从属一个父结点。或者类比为书本章节划分形成的章节嵌套,每个结点(文字段落)只允许从属于一个章节。树形结构的最高层次的结点称为根(root)结点,只有它没有父结点。如果放松关系r的约束,让结点可以从属于多个父结点,那么它就不是树形结构,而变为"图结构"了。从数学的角度看,树形结构和图结构的基本区别就是"每个结点是否仅仅从属于一个直接上级"。而线性结构和树形结构的基本区别是"每个结点是否仅仅有一个直接后继"。树形结构存在着很多变种,如二叉树结构、堆结构等,它们都有着各自独特的有效应用,在后续章节中将进行相关的分析讨论。

• **图结构(graph structure)**:有时称为网络结构,因特网的网页链接关系就是一个非常复杂的图结构。对于图结构的关系r没有任何约束,这样也就无法像线性结构及树结构那样,利用关系r的约束来设计图结构的存储结构。对于完全图而言,关系r的全部元组的数

量是很大的，它们的存储本身就是一个挑战性问题。不过在日常应用中，图结构往往只是层次结构的一种扩展——允许结点具有多个“直接上级结点”，关系 r 表现为树形结构约束的放松。

以上这种结构分类揭示出枯燥数据之间的相互关系，给出关系本身的一般性质，这对于理解数据结构以及正确设计算法都是重要的。当然，结点本身的数据类型及其含义也是极为重要的。后续各章还将对上述结构类别进行细分，并讨论各种具体数据结构类别的应用和相关的算法实现。

3. 结点和结构

对于数据结构 (K, R)，除了它的关系 r，结点数据类型也是非常需要关注的。它不限于基本数据类型，可以根据应用需要来灵活设计结点的数据类型。只要需要，完全可以将一个复杂的存储结构作为结点数据类型。因此，可以认为数据结构的设计是一层一层地进行的。根据求解问题的处理数据需要，先明确数据结点及其主要关系 r。在分析关系 r 的同时，也要分析其数据结点的数据类型。如果数据结点的逻辑结构比较复杂，那么把它作为下一个层次，再分析下一层次的逻辑结构。这是一种自顶向下的分析设计方法。

1.2.2 数据的存储结构

计算机的主存储器的特性可以简述如下：其存储空间提供了一种具有非负整数地址编码的、在存储空间上相邻的单元集合，其基本的存储单元是字节。计算机的指令具有按地址随机访问存储空间内任意单元的能力，访问不同地址所需的访问时间基本相同。数据的存储结构的一个主要任务，是要充分利用主存储器的“空间相邻”和“随机访问”两个特点，利用地址顺序单元空间相邻关系来表达数据结构的结点，每个结点通常被存储在一片连续地址的紧凑存储区域内。

为了表达逻辑关系 r，可以将它映射到存储结构的地址关系上，利用存储地址的顺序相邻关系，或者地址指针的指向关系来表达关系 r，其所采取的表达方法的好坏会非常影响存储空间的开销，而且也会影响到算法的效率。

可以用数学上的映射(mapping)来表示存储结构。数据的存储结构是建立一种由逻辑结构到存储空间的映射：对于逻辑结构 (K, r)，其中 $r \in R$，一方面，对它的结点集合 K 建立一个从 K 到存储器 M 的单元的映射：$K \to M$，其中每一个结点 $j \in K$ 都对应一个惟一的连续存储区域 $c \in M$。另一方面，还要把每一个关系元组 $(j_1, j_2) \in r$(其中 $j_1, j_2 \in K$ 是结点)映射为存储单元的地址顺序关系(或指针的地址指向关系)。

下面将分别介绍 4 种基本的存储映射方法。

1. 顺序的方法

用一块无空隙的存储区域存储结点数据称为顺序存储。顺序存储把一组结点存放在按地址相邻的存储单元里，结点间的逻辑关系用存储单元的自然顺序关系来表达的。简而言

之,利用一块存储区域,例如使用数组存储线性数据结构,数组的元素按照顺序存储方法存储。数组的元素是数据结点,结点之间的线性关系用地址单元的顺序关系来自然表达。

顺序存储法为使用整数编码访问数据结点提供了便利。例如,二维数组元素 $M[i][j]$ 的下标值(非负整数)i,j 就是该数组的一种索引,数组元素的存储地址可以根据下列公式用 i,j 进行计算,

$$\text{始址(元素 } M[i][j]) = \text{始址(元素 } M[0][0]) + (k \times i + j) \times (\text{元素尺寸})$$

其中,始址是存储区开始地址的简称,二维数组的存储方法一般是按行 $i = 0,1,\cdots,k-1$ 顺序存储其数组元素。数组元素所用存储单元大小是定长的,简称为(元素尺寸)。图 1.1 给出了二维数组 M 的存储区域 S。

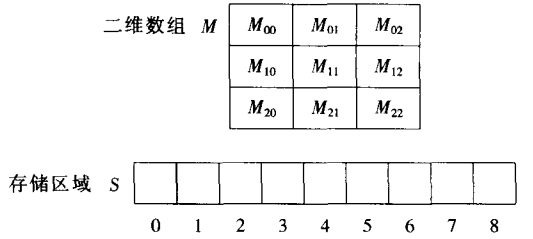


图 1.1 顺序存储法示意图

图 1.1 按照行向量顺序存储,在 S 中依次存放 3 个行向量。从地址计算来说,矩阵元素 M_{21}($M[2][1]$)的存储地址为 $S + (3 \times 2 + 1) \times$(元素尺寸)。

除线性结构外,非线性数据结构也可以部分地利用顺序存储方法,下面即将讨论的索引存储法,就是部分利用了顺序存储法来存储其索引指针。

顺序存储结构被称为紧凑存储结构,其紧凑性是指它的存储空间除了存储有用数据外,没有用于存储其他附加的信息。有些存储方法,如链接方法,就需要附加指针信息来表达其逻辑关系。紧凑性可以用"存储密度"来度量,"存储密度"即所存储的"有用数据"和该结构(包括附加信息)整个存储空间大小之比。显然,存储密度太小的存储结构是不好的。有时为了"用空间换取时间",在存储结构中存储一些附加信息还是很必要的。下述的链接方法、索引方法和散列方法都是牺牲存储空间来换取其他一些好处的,譬如用于提高算法的执行速度,或者让算法实现更为简便等。

2. 链接的方法

利用指针在结点的存储结构中附加指针字段的方法称为链接法。两个结点的逻辑后继关系可以用指针的指向来表达。一般而言,任意的逻辑关系 r 也可以使用这种指针地址来表达。一般的做法是将数据结点分为两部分,一部分存放结点本身的数据,称为数据字段;另一部分存放指针,称为指针字段,链接到某个后继结点,指向它的存储单元的开始地址。

多个相关结点的依次链接就会形成链索。

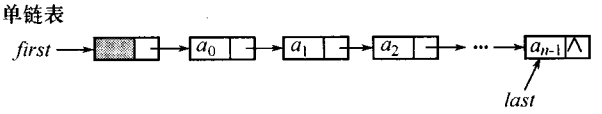


图 1.2　单链表示意图

图 1.2 中，*first* 指针指向单链表的首，在每个结点的存储结构中附加指针字段，其内部的指针指向后继结点。必要时，结点的指针字段允许存储多个指针，从而表达一个结点同时链接多个结点的情况。对于非线性结构会有多个后继结点的情况。

链接方法是经常使用的，特别是对于那些经常增删结点的复杂数据结构，顺序存储往往会遇到困难。链接方法结合 new 动态存储能够为这些复杂问题提供解决方法，但是它也有缺陷。为了访问结点集 K 中的某个结点，必须弄清楚该结点的存储指针。当不知道结点指针时，为了寻找符合条件的结点，就要在结点集 K 中沿着链索一个结点一个结点地比较搜索，所花费的时间是比较多的。

3. 索引的方法

索引法是顺序存储法的一种推广，它使用一个索引表存储一串指针，每个指针指向存储区域的一个数据结点。它的一个直接用途是可以把大小不等的数据结点顺序存储，并且仍然可以使用整数下标来间接访问数据结点位置。索引方法是要建造一个由整数域 Z 映射到存储地址域 D 的函数 Y:Z→D，把整数索引值 $z \in Z$ 映射到结点的存储地址 $d \in D$。函数 Y 称为索引函数，一般而言，它并不像数组那样，是简单的线性函数。在数据结点长度不等的情况下，索引函数就不是线性函数。

索引表 S 就是存储了一个索引函数，索引表的存储空间是附加在结点存储空间之外的。它的每一元素都是指向数据结点的指针(结点存储的开始地址)。因为索引表 S 是由等长元素(指针)组成的，所以索引表内的地址计算使用如下的线性计算公式：

$$\text{始址(元素 } S[i]) = \text{始址(元素 } S[0]) + i \times \text{(指针尺寸)}$$

通过上述公式，由索引号 i 可以计算出索引表中的单元 $S[i]$的始址，再通过读出 $S[i]$元素的内容(是指针)，访问真正需要访问的数据结点。

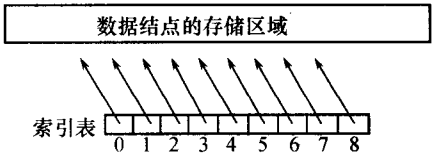


图 1.3　索引表的作用图示

用索引表指向存储区域中的各个结点，经过这个索引表可以把存储区域内结点的位置定位，如图 1.3 所示。

索引方法所付出的存储开销代价为其附加的指针存储空间。举例来说，若数据结点本身需要 10 个字节，而附加指针需要 4 个字节(存储器地址目前使用 32bit)，由此，其有效存储

密度(存储利用率)为 10/14,利用率不算高。不过,索引方法在程序设计中仍是一种经常使用的方法,对于非顺序的存储结构来说,使用索引表是快速地由整数索引值找到其对应数据结点的惟一方法。

4. 散列的方法

散列方法是索引方法的一种延伸和扩展,它利用散列函数(hash functions)进行索引值的计算,然后通过索引表求出结点的指针地址。散列函数 h 是整数函数,h: $S \rightarrow Z$,其中,S、Z 都是非负整数域的一个子集。即

对任意整数 $s \in S$,散列函数 $h(s) = z, z \in Z$

用一个简单的散列函数为例来说明,设字符串 s,它由 ASCII 字符 $s_1 s_2 s_3 \cdots s_L$ 组成。通过使用下述整数乘法"×"和求余数运算"%"的表达式,可以计算一个 $h(s)$:

$$h(s) = ((\cdots(int(s_1) \times int(s_2)\%N) \times int(s_3))\ \%N) \times \cdots \times int(s_L))\%N$$

其中 N 是一个正整数,而计算所得到的 $h(s)$为一个小于 N 的非负整数。这个 $h(s)$把长为 L 的任意字符串映射为一个小于 N 的非负整数。对一般的散列函数 $h(s)$,它把任意的二进制码 s(关键词)映射为一个小于 K 的非负整数,K 的大小取决于应用。

对一般的散列函数 $h(s)$,还要求满足一些重要的散列性质,在此将限于说明散列函数在数据存储结构的作用。以查字典为例,假定有一本数字化的"英文字典"可供快速查询有关英文词的解释。下述的散列法对这个问题很合适:设字典的词汇量不超过 K 个英文单词,用顺序存储方法存储一个"散列表",一共 K 个元素,其中每一个元素形如(w, d),w 是字典的英文单词(字符串),简称关键词,d 是 w 所对应的一个指针,指向对该单词 w 的字典解释文字。这样,当需要查询某个英文单词的时候,例如查询单词"rapid",就可以首先通过散列函数计算 h("rapid"),计算出来的非负整数值作为索引值,用于访问散列表得到其元素内容("rapid", d)。用查询的单词"rapid"和元素内容('rapid', d)的第一项"rapid"关键词比较,相同时就可以用其对应的指针 d 去访问并检索到该单词所对应的文字解释了。

在散列法存储表示里,关键的问题是恰当地选择散列函数,以及如何建造散列表,并研究在构建散列表时的"碰撞"问题等。这些问题将在第 9 章讨论检索方法时再详细地分析。

以上介绍的 4 种存储方法还可以组合起来,例如把顺序和链接结合起来等等。一个逻辑结构可以有多种不同的存储方案,在选择方案时,要根据其上的运算及其算法实现来确定。

1.3 抽象数据类型

抽象数据类型(Abstract Data Type, ADT)是描述数据结构的一种理论工具,其特点是把数据结构作为独立于应用程序的一种抽象代数结构,因此在很大程度上可以使人们独立于程序的实现细节来理解数据结构的主要性质和约束条件。程序是对数据结构进行函数变换,而为了保证程序的正确性,程序设计者必须注意相关的数据结构及其代数性质,注意所施加

变换应该满足的约束条件。

抽象数据类型不同于具体的数据结构，前者所描述的是一种模板，包括模板的结构和性质。而模板一般带有类型参数 T(元素的数据类型)，当它被具体的数据类型所代入时，就成为具体的数据类型。为了便于理解，下面以常见的二维矩阵类型 Λ = ***Matrix*** 为例，解释抽象数据类型，其元素类型 T 将作为矩阵模板的类型参数。

抽象数据结构 Λ 由 < 取值空间，运算集 > 两部分组成。

1. Λ 的取值空间

Λ 的取值空间即该数据结构的所有可能取值集合。举例来说，设数据结构 Λ 的组成元素的取值类型为 T。当 Λ 为二维矩阵 ***Matrix***(代表围棋棋盘)时，则它的元素位置代表棋盘的格子。数据类型 T 用于描述格子上棋子的黑白，它的取值集 Tz 为“黑”或“白”两个元素(1 个 bit)。令 $k \times k$ 是矩阵的大小，那么矩阵的可能取值集为 k^2 个 Tz 的卡式积，$Tz \times Tz \times Tz \times \cdots \times Tz$，需要用 k^2 bit 来表示。如果二维矩阵记录复数平面上的 $k \times k$ 个点，那么矩阵元素的类型 T 取复数类型 struct complex { double rPart; double iPart}，复数的数据类型由实部和虚部两部分组成，每一部分需要采用双字长浮点数 double，这样，二维矩阵的取值空间是 $k \times k$ 个复数类型元素，它们的可能取值是 $2\ k^2$ 个 Tz 的卡式积，$Tz \times Tz \times Tz \times \cdots \times Tz$。

可以看出，取值空间的大小和它的元素类型 T 密切相关。此外，对于取值空间，还需要说明它的大小是否动态改变。也就是说，该数据结构是否允许在程序运行中动态改变其取值空间大小。对于以上讨论的二维矩阵来说，取值空间一般不改变大小，它所代表的棋盘(或复平面的表示区域)的大小是固定的。而程序也就可以使用 array 类型，使用一个固定大小的存储空间来存储这种二维矩阵。当然，在程序中也经常使用动态数据结构。允许在程序运行中动态改变取值空间大小的数据结构称为动态数据结构。例如，变长字符串和变长的文本文件等就是动态数据结构类型。

2. Λ 的运算集

首先，使用变量名来访问数据结构可以看做为对它的一种基本运算。在程序中，访问(读/写)数据结构及其组成部分一般使用变量名，或使用几个变量名的组合。例如，访问二维矩阵 ***M*** 的某一行，用 ***M***[i]访问第 i 行，变量名 ***M*** 和整型下标变量名 i 的组合，前后用一对方括号是一种语法修饰。用 ***M***[2][3]访问 ***M***(2,3)，即矩阵的第 2 行、第 3 列的元素。这些基本访问方式对于理解一个数据结构的特点是极其重要的，它会影响数据结构的各种运算的时空开销。对于矩阵，矩阵的加法和乘法都是运算。二维矩阵对一维向量的乘法也是运算，结果是另一个向量。这些运算的实现算法内包括了对数据结构基本元素的读写访问。

一般使用函数来定义数据结构上的运算，参与运算的对象是函数的输入参数，而运算结果是其函数值。一般来说，一个数据结构的运算集并不是惟一的，也不是一成不变的，依赖于应用。此外，为了适合不同应用领域，它的元素类型 T 也可以不同，使得同一个结构的含义有所不同，并因此需要适当地添加新的运算种类。实际上，运算集的选择也会受到该数据