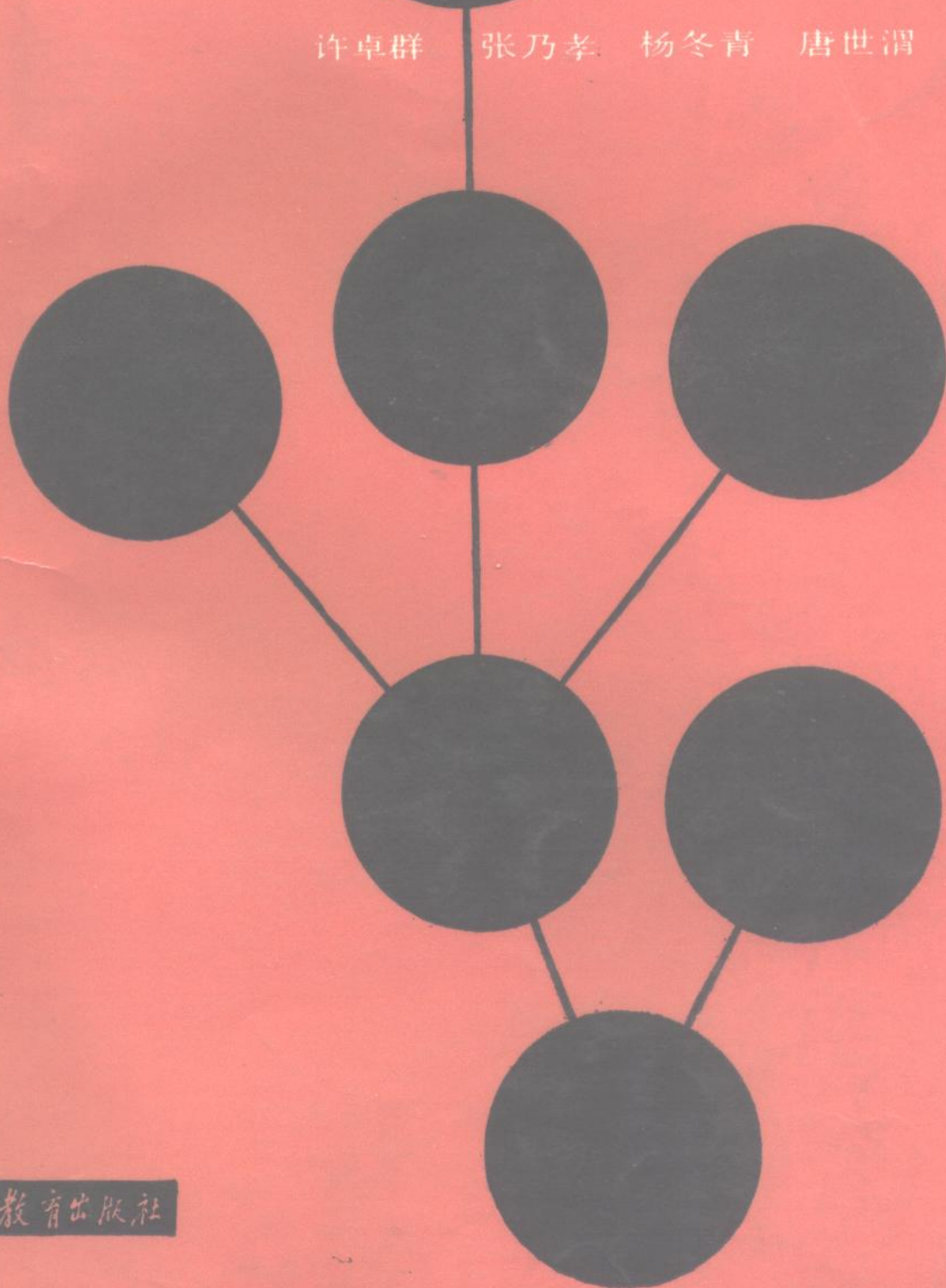


高等学校试用教材

数据结构

许卓群 张乃孝 杨冬青 唐世渭



12
2/1
高等教育出版社

高等学校试用教材

数 据 结 构

许卓群 张乃孝

杨冬青 唐世渭

高等教育出版社

内 容 提 要

本书是根据原教育部颁布的高等院校计算机软件专业数据结构课程教学大纲编写的教材。全书系统地介绍了各种常用的数据结构。内容丰富,概念讲解清楚,叙述严谨流畅,逻辑性强。书中对给出的每一种算法,均先描述了它的基本思路 and 要点,使得算法清晰易读,便于学生理解和掌握。有较丰富的例题和习题。

本书可作为高等院校计算机软件专业的教材或参考书,也可供广大从事计算机软件工作的科技人员自学参考。

本书由吉林大学庞云阶先生、刘大有先生,山东大学董继润先生审阅。

责任编辑:刘建元

高等学校试用教材

数 据 结 构

许卓群 张乃孝 杨冬青 唐世渭

*

高等教育出版社出版

新华书店北京发行所发行

北京第二新华印刷厂印刷

*

开本 787×1092 1/16 印张 20.5 字数 467 000

1987 年 5 月第 1 版 1988 年 2 月第 2 次印刷

印数 15 351—26 350

ISBN 7-04-001615-X/TP·20

定 价 3.50 元

前 言

《数据结构》是计算机软件的一门基础课程。计算机科学各领域及有关的应用软件都要用到各种数据结构。语言编译要使用栈、散列表及语法树；操作系统中用队列、存储管理表及目录树等；数据库系统运用线性表、多链表及索引树等进行数据管理；而在人工智能领域，依求解问题性质的差异将涉及到各种不同的数据结构，如广义表、集合、搜索树及各种有向图等等。数据结构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，并结合各种典型应用说明它们在进行各种运算(操作)时的动态性质及实际的执行算法。这样，不仅为读者学习后续软件课程提供了必要的知识准备，而且更重要的是进一步提高软件设计和编程水平。通过对不同存储结构和相应算法的对比以及上机编程练习，增强读者根据求解问题性质选择合理的数据结构并控制求解算法的空间、时间复杂性的能力。

本书是依照1983年原教育部部属高等院校计算机软件专业教学方案《数据结构》课程大纲编写的，要求读者对程序语言，如PASCAL语言具有必要的准备知识和上机经验。全书共分十八章及一个附录，大体上可看成为由四个部分组成，基本的线性结构及有关的典型应用是第一部分(第二章到第六章)；具有广泛应用价值的树形结构在第七章至第十一章讲述。这两部分占据了本书的主要篇幅。第十二章及第十三章介绍复杂数据结构，如图、稀疏矩阵及广义表等。一般来说，以上各章都是以假设数据全部在主存储器中存储(按字随机访问)为基础的。有关外存储器中的数据结构和文件组织放在第四部分，在第十四章到第十八章另作讨论，它们将涉及以页块为单位进行访问的外存储器和相应的常用数据结构。在内容上这一部分与后续的课程(操作系统、数据库系统)会有一定的重叠，讲授时可酌情取舍。

从教学改革的要求来看，作为必须讲授和最低教学要求部分，全书还有待进一步精选，书中相当一部分(如有*号的章节)可以只作为学生阅读参考之用，不作要求。应该鼓励学生分析和改进算法，提出新的算法，特别是实际编程实现各种算法。强调多做各章所附习题和上机练习的重要性。

本书是编者从一九八〇年以来为软件专业历届编写的数据结构讲义经多次修改而成的。由于水平所限，本书一定还存在不少问题，热诚希望读者指正。书中的各个实现算法虽经历届学生在课程中和毕业论文中进行过分析和上机检验，但还会存在软件故障性的错误，希望读者在发现后向我们指出。本书的最后修订稿是合作完成的：线性结构(第一部分)及第十三章由张乃孝同志执笔；树形结构(第二部分)及图(第十二章)由杨冬青同志执笔；第四部分及本书概论由唐世渭同志执笔。许卓群同志在讲义整理过程中对各章作了详细的修改。

许卓群 张乃孝 杨冬青 唐世渭

85年2月于北京大学

目 录

第一章 概论 1	§ 4 数据的存储结构..... 7
§ 1 为什么要学习数据结构..... 1	§ 5 数据的运算..... 9
§ 2 什么是数据结构..... 2	* § 6 数据结构的选择和评价..... 13
§ 3 数据的逻辑结构..... 3	习题..... 15

第一部分 线性结构

第二章 顺序表16	§ 3 串的运算.....80
§ 1 向量.....16	* § 4 模式匹配.....84
§ 2 栈.....21	习题.....92
* § 3 栈的应用——计算表达式的值.....24	第五章 内排序93
§ 4 栈与递归.....29	§ 1 基本概念.....93
§ 5 队列.....35	§ 2 插入排序.....94
* § 6 限制存取点的表.....39	* § 3 选择排序.....101
习题.....40	§ 4 交换排序.....103
第三章 链表与动态存储管理42	* § 5 分配排序.....108
§ 1 单链表.....42	§ 6 归并排序.....112
§ 2 栈和队列的链接存储表示.....45	习题.....115
§ 3 可利用空间表.....47	第六章 线性表的检索117
§ 4 线性表的其它链接存储表示.....50	§ 1 基本概念.....117
§ 5 存储管理问题概述.....57	§ 2 顺序检索.....117
§ 6 存储的动态分配和回收.....59	§ 3 二分法检索.....118
* § 7 伙伴(BUDDY)系统.....67	§ 4 分块检索.....121
习题.....75	§ 5 散列表的检索.....122
第四章 串 (STRING)76	§ 6 基于属性的检索.....132
§ 1 基本概念.....76	习题.....135
§ 2 串的存储表示.....77	

第二部分 树形结构

第七章 树形结构的概念137	§ 1 链式存储.....146
§ 1 树的概念.....137	§ 2 穿线树.....148
§ 2 二叉树的概念.....139	§ 3 顺序存储.....154
§ 3 树的二叉树表示.....140	习题.....159
§ 4 周游树形结构.....142	第九章 二叉树周游算法160
习题.....144	§ 1 使用栈的周游算法.....160
第八章 树形结构的存储146	§ 2 逆转链的周游算法.....162

* § 3 Robson 周游算法	164
* § 4 Siklóssy 周游算法	166
习题	168
第十章 树目录	169
§ 1 二叉排序树	169
§ 2 最佳二叉排序树	172
§ 3 平衡的二叉排序树	180
* § 4 字符树	186

习题	193
* 第十一章 树形结构的其它应用	194
§ 1 Huffman 算法及其应用	194
§ 2 堆排序	198
§ 3 决策树	203
§ 4 博弈树	206
习题	212

第三部分 复杂结构

第十二章 图	213
§ 1 图的概念	213
§ 2 图的存储表示法	216
§ 3 图的周游和生成树	219
§ 4 最短路径	224
§ 5 拓扑排序	229
* § 6 关键路径	233
习题	238

第十三章 多维数组、稀疏矩阵

和广义表	241
§ 1 多维数组	241
§ 2 稀疏矩阵	244
* § 3 稀疏矩阵的乘法	248
§ 4 广义表(LIST)的概念	251
§ 5 广义表的存储	253
§ 6 无用单元的收集与存储压缩	256
习题	260

第四部分 文件结构

第十四章 顺序文件	262
§ 1 外存储器简介	262
§ 2 文件结构概述	266
§ 3 顺序文件	269
习题	272
第十五章 散列(Hash)文件	274
§ 1 按桶(Bucket)散列	274
§ 2 可扩充散列	276
习题	281
第十六章 索引顺序文件	282
§ 1 静态索引结构	282
§ 2 动态索引结构	285

习题	295
* 第十七章 倒排文件	296
§ 1 倒排文件的存储结构	297
§ 2 倒排文件上的运算	299
习题	304
* 第十八章 外排序	305
§ 1 磁盘排序	305
§ 2 磁带排序	311
习题	314
附录 关于书写算法的若干规定	316
参考书目	319

第一章 概 论

在本章中将介绍有关数据和数据结构的概念,简述数据结构在计算机科学中的重要性,扼要说明本课程的内容。本章可以作为全书的导引,但有些概念可能在读完全书后才能有较深刻的理解。

§1 为什么要学习数据结构

我们常常听到人们把计算机称作数据处理机,从而可见数据在计算机领域中的重要地位。我们在这里所说的数据就是对现实世界的事物采用计算机能够识别、存储和处理的形式所进行的描述。简言之,数据就是计算机化的信息。计算机一问世,数据作为计算机程序的处理对象也就随之产生。在计算机发展的初期,由于计算机主要用于数值计算,处理的是数值数据,而且数据量小、结构简单、形式统一。所以当时程序工作者把主要精力放在程序设计技巧上,而并不重视如何在计算机中组织数据。但是,随着计算机软件和硬件的发展,计算机应用领域的扩大,数据的概念也被大大推广了,越来越多的非数值数据需要处理。据有人统计,现在,非数值计算占用了90%以上的计算时间。

计算机所处理的数据决不是杂乱无章的堆积,而是有着内在的联系,只有分析清楚它们的内在联系,对大量的数据才能进行有效处理。数据结构就是指数据间的相互关系。随着计算机应用的发展,不但处理的数据量越来越大,而且要处理的数据结构更为复杂。

关于什么是“计算机科学”,从六十年代以来一直有着争论,以Wegnor为代表的一种观点认为:“计算机科学”“包含一种关于计算图式的新的思想方法”。“工业革命中起核心作用的是‘能量’,在计算机革命中被‘信息’所取代”。他认为在计算机科学中具有核心作用的概念是“信息结构的转换”。计算机科学就是“一种关于信息结构转换的科学”。因而他认为构造有关信息结构转换的模型,对之进行研究,是计算机科学中带根本性的问题。而以Knuth为代表的另一种观点则认为:“计算机科学”“是算法的学问,算法是精确定义的一系列规则,指出怎样从给定的输入信息经过有限步骤产生所求的输出信息,‘算法’的特殊表示称为‘程序’,如同我们用‘数据’这个词来作为‘信息’的特殊表示一样”。我们且不去评论在计算机科学中到底是应该强调“信息结构”(或“数据结构”)还是应该强调“算法”(或“程序”),但无疑“信息结构”和“算法”应该是计算机科学研究的基本课题。而且应该指出,数据结构和算法之间存在着本质的联系,失去一方,另一方就没有什么意义。我们在谈到一种类型的数据结构时,总是离不开对这种类型数据结构需要施加的各种运算(又称操作,转换),例如,替换、插入、删除等等,而且只有通过对这些运算的算法的研究,才能更清楚地理解这种数据结构的意义和作用。反过来,当我们谈论一种算法时,也总是自然地联系到作为该算法处理对象和结果的数据。因此,在学习计算机科学中,数据结构的学习

起着基础性的作用。

§ 2 什么是数据结构

前一节中已提到数据结构是指数据之间的关系。但关于数据结构的概念至今并没有一个公认的标准定义,不过在谈到任何一种结构时,都自然地联系着对这种类型数据所需要的运算,以及为了在计算机上执行这些运算需要把这些数据如何存储在计算机中。所以数据结构概念一般包括:数据之间的逻辑关系,数据在计算机中的存储方式和数据的运算三个方面。

为了叙述上的方便,把上述数据结构的三个方面分别称为:数据的逻辑结构,数据的存储结构和数据的运算。例如,一个线性表,它的逻辑结构是指对下列一些问题的回答:哪一个元素是表中的第一个元素?哪一个元素是表中的最后一个元素?哪些元素在一个给定元素之前或之后?等等。它的存储结构是指这样的内容:它的元素在存储器中是顺序地邻接存放还是用指针连在一起的?等等。数据运算可以包括:在表中找一个元素,从表中删去一个元素,向表中插入一个元素,等等。希望读者学习时,不要孤立地去理解一个方面,而要注意它们之间的联系。

例1 某工厂职工工资表形式为:

编 号	姓 名	基本工资	附 加 费		代 扣 费		实发金额
			交通补贴	夜班费	房租	伙食	
1	王 一	62	2.8	0	3.3	20	41.5
2	丁 二	43	0	0	0.9	15	27.1
3	李 四	130	2	0	10	30	92
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

其中王一、丁二、李四等每个职工在表中各占一行,每行的信息说明一个职工的工资情况,是工资表的基本单位。我们称整个工资表为一个数据结构,而把表中的每一行称为一个结点。结点是数据结构中讨论的基本单位(有些情况下也把结点称作元素、记录、表目等)。上例表中每一行由编号、姓名、基本工资、附加费、代扣费、实发金额等数据项组成;而附加费和代扣费两项又分别由交通补贴、夜班费和房租、伙食等项组成,我们把编号、姓名、基本工资、实发金额等数据项称为初等项,初等项由一个或多个字符组成,是有独立含义的最小标识单位。我们把附加费、代扣费称为组合项,组合项由一个或多个初等项或组合项组成,是有独立含义的标识单位。结点由一个或多个组合项组成,构成结点的组合项又称为字段,我们把其值能唯一确定一个结点的字段或字段组合称为关键码,在上例中编号字段就是关键码,一个职工编号就唯一确定一个职工,即唯一确定工资表中的一行。在PASCAL语言中,记录相当于结点,域相当于字段,而在本书的叙述中,一般采用结点和字段这样的术语。

上述工资表就是一张线性表,表中结点和结点之间是一种简单的线性关系,表中有且只有一个结点为表头(第一个结点),有且只有一个结点为表尾(最后一个结点),对表中任一结点,与它相邻且在它前面的结点最多只有一个,这就是上述工资表的逻辑结构。当上述工资表存入计算

机时, 假定结点和结点之间采用顺序存放的方式, 这就是存储结构. 对这样的表格, 在职工调离时相应的结点要删除, 在招收新职工时, 要增加新的结点, 调整工资时要修改. 究竟如何进行插入、删除、修改, 这就是数据的运算问题. 弄清楚以上这些问题, 工资表的结构就完全明白了.

综上所述, 按某种逻辑关系组织起来的一批数据, 按一定的存储表示方式把它存储在计算机的存储器中, 并在这些数据上定义了一个运算的集合, 就叫做一个数据结构.

§ 3 数据的逻辑结构

对数据间关系的描述是数据的逻辑结构, 形式地可以用一个二元组

$$B = (K, R)$$

来表示, 其中 K 是结点的有穷集合, 即 K 是由有限个结点所构成的集合; R 是 K 上的关系的有穷集合, 即 R 是由有限个关系所构成的集合, 而其中的每个关系都是从 K 到 K 的关系. 在下面的叙述中, 凡不易产生混淆之处, 我们有时就把数据的逻辑结构简称为数据结构.

一、结点的类型

现在我们从结点的值的组成方式及数据类型的角度, 来讨论结点的分类. 设 K 是结点的有穷集合且每个结点 $k (k \in K)$ 的值(即数据)可以由 n 个字段组成, 第 i 个字段记为 w_{ik} , 所以 k 可以表示为

$$k = (w_{1k}, w_{2k}, \dots, w_{nk}), n \geq 1$$

其中每个 w_{ik} 是一个组合项, 它可以由一个或多个初等项和组合项组成, 也可以就是一个初等项.

结点可以分为两大类型: 初等类型和组合类型. 只包含一个初等项的结点属于初等类型, 否则是组合类型.

有五种基本的初等类型:

整数类型(integer): 取值为计算机可以表示范围内的整数.

实数类型(real): 取值为计算机可以表示范围内的实数.

布尔类型(boolean): 取值为真(true)和假(false).

字符类型(char): 取值为计算机可以表示的字符集的元素.

上述四种基本的初等类型都是计算机能直接存储并能直接用机器指令对它们进行处理的.

另一种初等类型即**指针(pointer)**. 该类型的值一般来说对应于存储器里的一个地址或相对地址. 指向结点的指针是指向该结点的第一个单元的地址或者指向该结点在结构中的相对位置. 当指针值存入计算机时形式上与整数相似, 但由于指针的含义代表的是地址, 通常不能用来做算术运算(特别是乘除法), 也不能取负值.

组合类型是由初等类型用某种方式(可以是很简单的也可以是很复杂的)组合而成的. 不同的组合方式形成不同的类型, 例如, 数组类型、记录类型等. 上节所举工资表的例子中结点就属组合类型.

早期的高级语言例如 FORTRAN, ALGOL60 等都引进了基本初等类型, 在这些语言中采用“类型说明”的方式指出语言中使用的变量的类型。较新的高级语言, 例如, ALGOL68, PASCAL 等把类型说明的功能大大扩充, 允许用户自己定义更多的初等类型和组合类型。

二、结构的分类

设 $B = (K, R)$ 是一个逻辑结构, r 是一个 K 到 K 的关系, $r \in R$ 。若 $k, k' \in K$, 且 $\langle k, k' \rangle \in r$, 则称 k' 是 k 的后继, k 是 k' 的前驱。这时 k 和 k' 是相邻的结点(都是对 r 而言)。如果不存在一个 k' 使 $\langle k, k' \rangle \in r$, 则称 k 为关于 r 的终端结点(即无后继的结点)。如果不存在 k' 使 $\langle k', k \rangle \in r$, 则称 k 为关于 r 的开始结点(即无前驱的结点)。如果 k 既不是终端结点, 也不是开始结点, 则称 k 是内部结点。

在本书所讨论的数据结构中, 一般只讨论包含一个关系 r 的关系集 R , 即 $R = \{r\}$ 。对于 R 中包含多个关系的情况, 可以用类似的方法进行讨论。

数据结构分为线性结构和非线性结构。在线性结构里有且仅有一个终端结点和一个开始结点, 并且所有的结点都最多只有一个前驱和一个后继, 线性表就是典型的线性结构。在线性表的逻辑结构 $B = (K, R)$ 中, 若 K 是包含 n 个结点 $\{k_1, k_2, \dots, k_n\}$ (即表中有 n 个表目) 的集合, $R = \{r\}$, 关系 r 是一组有序对 $\langle k_{i-1}, k_i \rangle$ 构成的集合, 即

$$r = \{\langle k_{i-1}, k_i \rangle \mid k_i \in K, 1 \leq i \leq n\}$$

即 K 里所有的结点都可以按 r 排成一个序列 $k_1, k_2, \dots, k_n$, 其中 k_1 为开始结点, k_n 为终端结点。显然 K 中的每个结点最多只有一个前驱和一个后继, 在第一部分中我们将详细讨论线性表。非线性结构中最重要的是树结构, 在树的逻辑结构 $B = (K, R)$ 中, K 是包含 n 个结点的集合, $R = \{r\}$, 而 r 满足下列条件: 有且仅有一个称为根的结点没有前驱; 其他结点有且仅有一个前驱; 对于非根结点都存在一条从根到该结点的路径 (即: 若根为 w , 该结点为 k , 则存在一个结点序列 $k_0, k_1, \dots, k_s$ 其中 $k_0 = w, k_s = k$, 使得 $\langle k_{i-1}, k_i \rangle \in r, 1 \leq i \leq s$)。关于树结构在第二部分中重点讨论。最一般情况是图结构, 在图的逻辑结构 $B = (K, R)$ 中, K 是包含 n 个结点的集合, 对于 K 中结点相对于关系 $r (r \in R)$ 的前驱和后继的个数都不作限制。第三部分介绍图结构和其它复杂结构。外存储器上的数据结构——文件, 在第四部分讨论, 从本质上讲, 文件是一种线性结构, 与线性表相似, 而文件的索引一般都与树结构相联系。

三、结点和结构

结点和结构是相对而言的, 这有些类似于元素和集合的关系。我们可以不严格地说, 结构是由若干结点及其相互之间的关系所组成。在一定场合下, 当我们需要研究某结点的内部组成情况时, 我们就可以把该结点看作一个结构, 而把组成该结点的数据项看作结点, 把数据项组成该结点的组成方式看作结点之间的关系。一般说来, 一个结点可以包含 n 个字段, 每个字段又可以是一个组合项, 一个组合项又可以由多个组合项组成, 如此等等。所以实际上一个结点常常是若干个初等项按某种规则组合起来的, 这种组合的方式可以是线性的, 也可以是非线性的。例如, 当

我们关心数组和数组之间的关系时，可以把数组定义为结点；但当我们关心数组内各元素之间的关系时，我们又可以把数组元素看成为结点，而数组就成了我们讨论的数据结构。在另一些情况下，我们还可以把一个二维数组看成是向量的向量，从而简化讨论的关系。这时向量就成了我们结构中的结点。

另外，描述结点的前驱和后继关系的指针，一般都作为结点的一个或若干个字段，从这一点来讲，结点之间的关系是和结点的值一起存放在计算机中的，它们相对计算机而言都是机器处理的对象，即都是某种类型的数据。

由于数据结构讨论的基本单位是结点，所以很多情况下并不重视结点的构成，而是把结点抽象地看作一个整体，重点来讨论结点之间的关系。

在以后各章中，当需要描述结点的组成时，我们采用类似 PASCAL 语言中的类型说明的方法。

例 2 某系学生成绩登记表。每个学生的有关数据是一个结点，它由学号、姓名、专业、考试成绩四个字段组成，其中考试成绩是一个组合项，它又包括数学、物理、程序设计、政治和英语五个成绩的初等项。采用 PASCAL 的说明表示如下：

```
TYPE  score = 0..100;
      exam = ARRAY [1..5] OF score;
      student = RECORD
          no: integer;
          name: string;
          specialty: string;
          c: exam
      END;
```

其中 score, exam, student 是类型标识符，score 称为子界类型，取值是从 0 到 100 的整数，用 0..100 来说明，exam 称为数组类型，由五个类型为 score 的元素组成，数组元素的下标是 1 到 5 之间的整数。student 称为记录类型（在本例中它对应一个结点），由四个域（相当于字段）组成，no, name, specialty 和 c，即为域标识符，分别表示学号、姓名、专业和考试成绩。类型说明不仅描述了结点的组成方式，同时还给出了组成结点的各字段的名称（域标识符）。域的类型也在类型说明中描述。integer 表示整数，string 表示字符串。但是必须注意，类型说明仅给出一类变量的取值范围，并没有指出哪些变量属于该类型，更没有给出变量具体的值。只有当用变量说明把某变量的标识符与该类型标识符联系在一起时，才确定了变量的类型，而通过给变量赋值才能使变量定值。例如，在例 2 中我们加入变量说明

```
VAR A1, A2, A3: student
```

就定义了三个变量 A₁, A₂, A₃，它们都具有同类型说明中 student 那样的类型。当我们执行赋值语句

```
A1. name ← 'ZHANG MIN'
```

和

$A_2.c[3] \leftarrow 95$

以后,就给变量 A_1 的 name 字段赋值为 ZHANG MIN, 给 A_2 的 c 字段的下标为 3 的初等项赋值为 95.

有关 PASCAL 语言的详细规定可参看 PASCAL 的正式文本.

为了直观,有时也用图示法来表示一个逻辑结构. 表示法是: 对应每一个结点 K , 用一个长方形框表示, 里面写上 K , 当关心 K 的值时, 可以按字段标出字段值, 长方框之间用有向线段联系. 表示结点之间的前驱、后继关系, 对同一个结点集合上的不同关系在图上用不同的线段来区分.

例 3 一批数据的逻辑结构 $B = (K, R)$, 其中

$$\begin{aligned} K &= \{k_1, k_2, \dots, k_9\}, \quad R = \{r\}, \\ r &= \{\langle k_1, k_2 \rangle, \langle k_1, k_3 \rangle, \langle k_1, k_4 \rangle, \\ &\quad \langle k_1, k_7 \rangle, \langle k_1, k_8 \rangle, \langle k_4, k_5 \rangle, \langle k_4, k_6 \rangle, \langle k_8, k_9 \rangle\} \end{aligned}$$

可用图 1-1(a)表示.

例 4 一批数据的逻辑结构 $B = (K, R)$, 其中

$$\begin{aligned} K &= \{k_1, k_2, \dots, k_{10}\}, \quad R = \{r_1, r_2\}, \\ r_1 &= \{\langle k_1, k_2 \rangle, \langle k_1, k_8 \rangle, \langle k_2, k_3 \rangle, \langle k_2, k_7 \rangle, \\ &\quad \langle k_3, k_4 \rangle, \langle k_3, k_5 \rangle, \langle k_3, k_6 \rangle, \langle k_8, k_9 \rangle, \langle k_8, k_{10} \rangle\}, \\ r_2 &= \{\langle k_{10}, k_4 \rangle, \langle k_4, k_2 \rangle, \langle k_2, k_3 \rangle\} \end{aligned}$$

可用图 1-1(b)表示, 其中实线表示关系 r_1 , 虚线表示关系 r_2 .

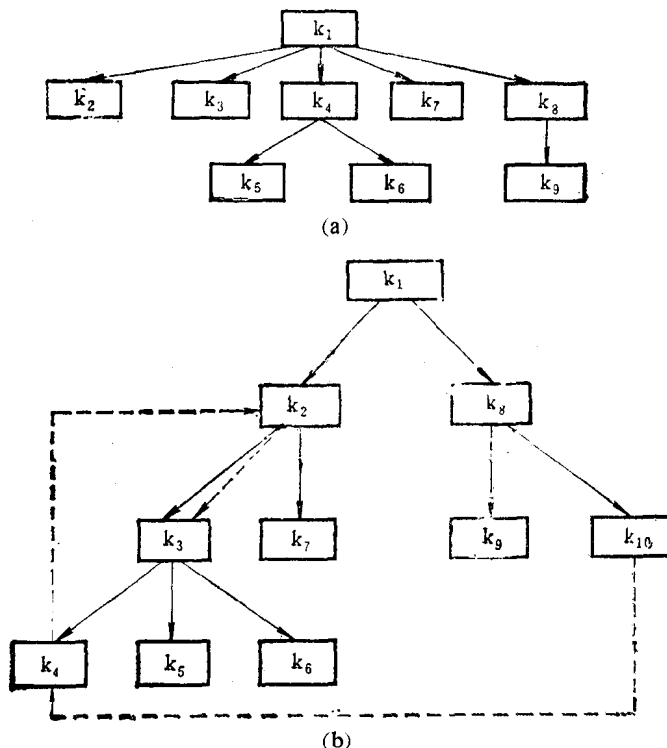


图 1-1 逻辑结构的图示法

§ 4 数据的存储结构

数据的逻辑结构是从逻辑关系上来观察数据,它与数据的存储无关,是独立于计算机的。数据的存储结构是逻辑结构在计算机存储器里的实现,它是依赖于计算机的。

计算机的存储器(主存)是由有限多个存储单元组成的,每个存储单元有唯一的地址,各存储单元的地址是连续编码的,每个存储单元 Z 都有唯一的后继单元 $Z' = \text{suc}(Z)$ 。 Z 和 Z' 称为相邻单元。一片相邻的存储单元的整体叫做存储区域,记作 M 。在不产生混淆的情况下, Z 可以用来表示一个存储单元,也可以用来表示该存储单元的地址。地址 $Z' = \text{suc}(Z) = Z + 1$ 。

设有逻辑结构 $B = (K, R)$, 要把 B 存储在计算机中, 首先必须建立一个从 K 的结点到 M 的单元的映象 $S: K \rightarrow M$, 即对于每一个 $k \in K$ 都有唯一的 $Z \in M$ 使得 $S(k) = Z$, Z 为 K 结点所占存储空间中的始单元。同时这个映象应具有明显地或隐含地体现关系 R 的能力。

由于计算机的存储区域总是有限的,所以如何合理地使用它,使得有限的存储区域发挥最大的作用,这是存储管理问题,我们在第三章中将对这个问题进行讨论。

我们用 $\text{LOC}(k)$ 表示结点 k 对应的存储单元的地址。

有四种基本的存储映象方法,下面分别加以介绍。

1. 顺序的方法

这种方法主要用于线性的数据结构,它把逻辑上相邻的结点存储在物理上相邻的存储单元里,结点之间的关系由存储单元的邻接关系来体现。如果结点 k 所占存储空间的第一个单元为 Z , 即 $\text{LOC}(k) = Z$, 而最后一个单元为 Z_1 , 那么 Z_1 的后继单元 $Z' = \text{suc}(Z_1)$ 就是 k 的后继 k' 的第一个存储单元。

例 5 $B = (K, R)$, $K = \{k_1, k_2, k_3, k_4, k_5\}$, $R = \{r\}$,

$$r = \{\langle k_1, k_2 \rangle, \langle k_2, k_3 \rangle, \langle k_3, k_4 \rangle, \langle k_4, k_5 \rangle\}$$

假定每个结点占一个存储单元, 结点 k_1 存放在 200 号单元中, 则顺序存储实现如图 1-2 所示。

这里存储区域被结点的值稠密地填满了。
需要注意的是,在一般情况下,每个结点所占的存储单元并不只一个,而且所占单元个数也可以不一定相等,这时顺序存储存放时就不那么整齐,但仍然是一个接着一个地填满了整个区域。

200	k_1
201	k_2
202	k_3
203	k_4
204	k_5

图 1-2 顺序存储的线性数据结构

另外对非线性的数据结构也可以采用局部线性化的方法实现顺序存储。例如,在树形结构中可以把结点按某种规则排成序列,用顺序存储方法把结点内部的信息稠密地存放在一起,而对结点之间的关系采用其它的存放方法。具体做法以后介绍。

2. 链接的方法

这种方法是给结点附加上指针字段。即是将结点所占的存储单元分为两部分,一部分存放

结点本身的信息,称数据项;另一部分存放此结点的后继结点所对应的存储单元的地址,称指针项。指针项可以包括一个或多个指针,以指向结点的一个或多个后继,或记录其他信息(当然,一般来说,指针是用来表示某结点的地址的,并不是只能表示后继结点的地址)。

我们用 info 表示结点的数据项,用 link 表示结点的指针项。那么,如果 k' 是 k 的后继结点,就有

$$LOC(k') = k.link$$

前面例 5 的逻辑结构可用链接的方法存储如图 1-3 所示。

其中记号 Λ 表示空指针,即该指针不具有有意义的值(不表示任何具体结点的单元地址)。

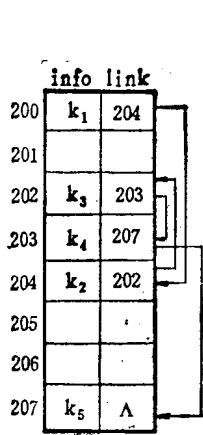


图 1-3 链接存储的线性数据结构

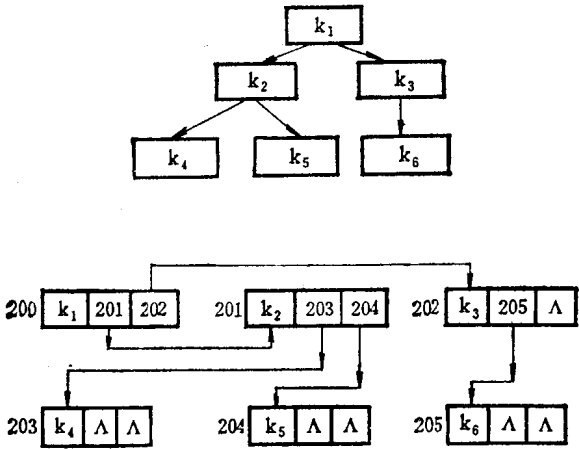


图 1-4 一个非线性数据结构的图示和链接存储

例 6 逻辑结构为 $B = (K, R)$, 其中

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

$$r = \{\langle k_1, k_2 \rangle, \langle k_1, k_3 \rangle, \langle k_2, k_4 \rangle, \langle k_2, k_5 \rangle, \langle k_3, k_6 \rangle\}$$

用链接的方式来实现这个逻辑结构, 因为有些结点有两个后继, 所以让指针项包括两个指针, 分别指向两个后继。

我们可以看到, 在顺序的存储表示中所有的存储空间都被结点数据占用了, 而在链接的存储表示中就不是这样, 有一部分存储空间里存放的是表示关系的附加信息——指针。

如果所有的存储空间都分配给了数据, 则这个存储结构叫紧凑结构, 否则叫非紧凑结构。显然顺序的存储表示是紧凑结构, 链接的存储表示是非紧凑结构。

结构的存储密度定义为数据本身所占的存储量和整个结构所占的存储量之比, 即

$$d = \frac{\text{数据本身所占的存储量}}{\text{整个结构所占的存储总量}}$$

紧凑结构的存储密度为 1, 非紧凑结构的存储密度小于 1。存储密度越大, 则存储空间利用效率越高。但是存储附加的信息会带来运算上的方便。例如, 在链表里, 因为存储了指针, 所以链表比起顺序表来作插入、删除运算要方便得多。牺牲了存储空间, 换取了机器时间, 这在以后还要详细讲到。

3. 索引的方法

在线性的结构里,结点可以排成一个序列: $k_1, k_2, \dots, k_n$, 每个结点 k_i 在序列里都有对应的位置数 i , 这个位置数就是结点的索引. 索引的存储结构就是用结点的索引号 i 来确定结点的存储地址. 有两种实现方法:

(1) 建立附加的索引表,索引表里第 i 项的值就是第 i 个结点的存储地址.

(2) 当每个结点所占单元个数都相等时,可用位置数的线性函数的值来指出结点对应的存储地址,即第 i 个结点 k_i 的地址是

$$\text{LOC}(k_i) = (i-1) \cdot p + q$$

其中, p 为结点所占单元个数, q 为开始结点 k_1 对应的存储地址.

例如,一维数组 $A[N]$, 设 $A[1]$ 的存储地址是 d , 每个数组元素占两个存储单元, 那么 $A[i]$ 的存储地址为

$$\text{LOC}(A[i]) = 2 \cdot (i-1) + d$$

4. 散列的方法

这种方法的主要思想是根据结点的值来确定它的存储地址.

在结点 k 的字段里取一个(或几个)字段的值 w_{ik} 作为关键码, 结点 k 对应的存储地址由函数 f (称为散列函数)确定:

$$\text{LOC}(k) = f(w_{ik})$$

所以这个方法也叫关键码——地址转移法. 用散列法进行存储表示时, K 的结点 k 分散地存储在 M 的存储单元里.

在散列法存储表示里关键的问题是选择散列函数和研究解决碰撞的办法, 这在第二章讲线性表时再详细讨论.

以上介绍的是四种基本的存储映象方法, 这些基本的方法还可以组合起来对数据结构进行存储映象. 例如表的存储, 当表的项不超过一个给定的极限 l 时, 表可以顺序地存储在连续的存储区域里, 当超过此极限时, 由一个指针指向溢出区域, 超过的项就放在这个附加的溢出区域里, 这就是顺序和链接相结合的存储方法.

一般数据结构的存储映象都是这四种基本映象之一, 或是它们的组合. 同一个逻辑结构可以有几种不同的存储映象方法, 选择哪一种要根据运算的方便来具体确定.

存储结构是数据结构的三个方面之一, 若逻辑结构相同但存储结构不同, 则为不同的数据结构, 有时这种不同是相当大的, 以致于人们用不同的数据结构名称来标识它们. 例如, 线性表是一逻辑结构, 若采用顺序方法的存储表示, 则人们称该数据结构为顺序表; 若采用链接方法的存储表示, 则人们称该数据结构为链表.

§5 数据的运算

为了更有效地处理数据, 提高数据运算效率, 我们按一定的逻辑结构把数据组织起来, 并选择适当的存储表示方法把按逻辑结构组织好的数据存储到计算机的存储器里. 数据的运算是

定义在数据的逻辑结构上的,但运算的具体实现要在存储结构上进行。数据的各种逻辑结构有相应的各种运算,每种逻辑结构都有一个运算的集合,这里只列举几种常用的运算,作一简要介绍。

1. 检索

检索就是在数据结构里查找满足一定条件的结点。一般是给定一个某字段的值,找具有该字段值的结点。

2. 插入

往数据结构里增加新的结点。

3. 删除

把指定的结点从数据结构里去掉。

4. 更新

改变指定结点的一个或多个字段的值。

插入、删除、更新运算都包含着一个检索运算,以确定插入、删除、更新的确切位置。

5. 排序

保持线性结构的结点序列里结点数不变,把结点按某种指定的顺序重新排列。例如,按结点中某一字段的值由小到大对结点进行排列。又如字符串的序列按字典顺序排列,整数序列按上升顺序排列等等。排序是一种非常有用的运算,第三章中我们将介绍排序的若干算法。

一个算法是规则的有穷集合,这些规则为解决某一特定类型问题规定了一个运算序列,此外一个算法应该具有下列特性:

1. 有穷性。一个算法必须总是在执行有穷步之后结束。

2. 确定性。算法的每一步,必须是确切地定义的。对于每种情况,有待执行的动作必须严格地和清楚地规定。

3. 输入。一个算法有零个或多个输入。它们是在算法开始前对算法最初给的量。这些输入取自于特定的对象集合,例如取自于整数集。

4. 输出。一个算法有一个或多个输出。它们是同输入有某种特定关系的量。

5. 可行性。算法应该是可行的。这意味着算法中所有有待实现的运算都是基本的,即是说,它们原则上都是能够由人们仅用笔和纸做有穷次运算即可完成的。

算法的含义与程序十分类似,但也有些差别。一般来说,一个程序并不需要满足上述的第一个条件(有穷性),例如操作系统,只要整个系统不遭破坏,操作系统就永不结束。另外程序是用机器可执行的语言书写的,而算法通常并没有这种限制。

在这里我们举一个例子,书写一个关于排序的简单算法,但其目的并不是为了讲解排序算法的本身,而是为了叙述书写算法的规定和怎样书写算法,使读者在怎样设计算法方面有个印象。

假设有 $n \geq 1$ 个不同的整数 $a_1, a_2, \dots, a_n$ 组成一个数列,现在要求对这个数列中的 n 个整数按从小到大进行排序。

我们把整个数列作为一个结构,每个整数看作一个结点,把整数在数列中的位置顺序看作结

点之间的关系。显然这是一个线性结构。

存储结构我们选择数组 A 来存放这个数列，把整数 a_i 存放在数组第 i 个元素位置 $A[i]$ ($1 \leq i \leq n$) 中。

现在考虑排序的算法，先写出粗略的思路：

1. 从所有整数中选出一个最小的整数放到一个已排好序的数列中，作为该数列中的第一个数。
2. 从剩下的未排序的整数中选出最小的整数放到已排好序的数列中，接在前一个数的后面。

反复执行步骤 2，直到所有整数都放到排好序的数列中。

实际上步骤 1 是步骤 2 的特殊情况，若用步骤 2 来代替步骤 1，则“剩下的未排序的整数”即为“所有整数”，而已排好序的数列此时为空。因而步骤 1 可以取消，只要重复执行 n 次步骤 2 即可。

现在把这一粗略的思路进行细化。我们可以认为：共执行 n 次重复。 i 从 1 开始，每执行一次重复， i 就加 1。当执行完第 $i-1$ 次重复时， $A[1]$ 到 $A[i-1]$ 为已排好序的数列，而 $A[i]$ 到 $A[N]$ 为剩下的未排序的整数。我们从 $A[i]$ 到 $A[N]$ 中选出最小整数放在 $A[i]$ 处。这样可以写出稍细一些的思路：

1. i 以 1 为步长。从 1 到 n ，循环执行
 - (1) 检查 $A[i]$ 到 $A[N]$ ，并选出最小的整数，设为 $A[j]$ 。
 - (2) 把 $A[i]$ 与 $A[j]$ 进行交换。
2. 算法结束。

在上述思路中，实际上第 n 次重复不必执行，因为此时只剩下了一个未排序的整数，因而不必再执行选出最小整数等操作了。现在尚需对上述思路中的步骤 1. (1) 和 1. (2) 进一步细化。显然，步骤 1. (2) 是容易实现的，只要引进一个中间变量，暂时存放 $A[i]$ 即可；而步骤 1. (1) 可以这样来完成：先把 $A[i]$ 作为最小整数，然后把 $A[i]$ 与 $A[i+1]$, $A[i+2]$, ... 进行比较，一旦出现比 $A[i]$ 小的整数，譬如说 $A[k]$ ，则就把 $A[k]$ 作为新的最小整数，然后再把 $A[k]$ 与 $A[k+1]$, $A[k+2]$, ... 进行比较等等。这样我们可以写出此例的最后算法。其变量说明为

VAR A:ARRAY[1..n]OF integer;

i, j, k, t:integer;

其算法如下：

算法 1.1 整数排序

1. 循环 i 以 1 为步长，从 1 到 $n-1$ ，执行
 - (1) $j \leftarrow i$ [把 $A[i]$ 作为最小整数]
 - (2) 循环 k 以 1 为步长，从 i 到 n ，执行
 - 若 $A[k] < A[j]$
 - 则 $j \leftarrow k$ [从 $A[i]$ 到 $A[N]$ 中选出最小整数]