21世纪高等学校计算机规划教材

21st Century University Planned Textbooks of Computer Science

数据结构

Data Structure

宗大华 宗杰 黄芳 编著

- 涵盖最新计算机考研大纲内容
- 从算法描述、分析和讨论三方面进行全方位讲述
- 示例、习题内容丰富全面

精品系列

人民邮电出版社
POSTS & TELECOM PRESS

21世纪高等学校计算机规划教材

21st Century University Planned Textbooks of Computer Science

数据结构

Data Structure

宗大华 宗杰 黄芳 编著

精品系列

人民邮电出版社

北京

图书在版编目（CIP）数据

数据结构 / 宗大华编著. -- 北京 : 人民邮电出版社, 2010.11
21世纪高等学校计算机规划教材
ISBN 978-7-115-22998-4

Ⅰ. ①数… Ⅱ. ①宗… Ⅲ. ①数据结构－高等学校－教材 Ⅳ. ①TP311.12

中国版本图书馆CIP数据核字(2010)第133060号

内 容 提 要

“数据结构”是高等院校计算机学科的一门专业基础课，其内容对学习后继课程有重要意义，对程序设计有实用价值。

本书内容分为3个部分：第1部分是第1章，它对“数据结构”做了概要性说明；第2部分包括第2章～第7章，具体涉及线性表、堆栈、队列、串、数组、矩阵、广义表、二叉树、树和森林、图等内容；第3部分由第8章和第9章组成，是对各种数据的查找和排序方法的介绍。

本书语言明快、流畅，概念描述准确、清晰，算法介绍全面、详实，各章都安排有大量的例子和习题，有助于教师备课和学生自学。

本书可作为高等院校计算机及相关专业本科生“数据结构”课程的教材，也可作为从事各种程序设计和计算机应用工作的读者的参考书。

21世纪高等学校计算机规划教材

数 据 结 构

◆ 编　著　宗大华　宗　杰　黄　芳
责任编辑　刘　博

◆ 人民邮电出版社出版发行　北京市崇文区夕照寺街14号
邮编　100061　电子函件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
中国铁道出版社印刷厂印刷

◆ 开本：787×1092　1/16
印张：25　2010年11月第1版
字数：660千字　2010年11月北京第1次印刷

ISBN 978-7-115-22998-4

定价：39.00元

读者服务热线：(010)67170985　印装质量热线：(010)67129223
反盗版热线：(010)67171154

前　言

“数据结构”是高等院校计算机学科的一门专业基础课，它涉及的内容不仅对学习各后继课程（如操作系统、编译原理、数据库原理等）有着重要的意义，就是在平常的程序设计中，也有很广泛的实用价值。

随着计算机应用的逐步深入和发展，数值的和非数值的数据都成为了计算机系统加工、处理的对象。为此，必须分析数据间的关系，弄清数据间的内在联系，“数据结构”就此应运而生，它关注和探究3个问题：第一，以用户的角度去观察这些“关系”和“联系”，从中抽象出数据的各种“逻辑结构”；第二，以数据存放到计算机里体现出这些“关系”和“联系”，去实现数据的各种“物理（存储）结构”；第三，基于物理结构，可以对数据进行的各种基本加工和处理（查找、插入、删除、遍历等）算法。

本书的内容就是基于以上3个需要关注和探究的问题，进行讨论和展开的。可以把它们划分成3个部分：第1部分是第1章，对“数据结构”做了概要性的说明；第2部分是对各种数据结构的介绍，由第2章～第7章组成，具体涉及线性表、堆栈、队列、串、数组、矩阵、广义表、二叉树、树和森林、图等内容；第3部分是对各种数据的查找和排序方法的介绍，由第8章和第9章组成。

“数据结构”是一门在教学计划中安排得很靠前的课程，是一门具有承上启下作用的课程，是一门理论性和抽象性很强的课程，也是一门应用性和实践性极为突出的课程。正是由于这样的一些特点，本书在整个编写过程中做了如下几个方面的努力。

1. 本书内容涵盖《2009计算机考研大纲》中关于“数据结构”部分提出的要求。

2. 在引入新概念或讲述新算法时，尽可能用简洁、清晰的语言进行表述，安排足够的例子做直观的解释和说明，以便能够帮助读者对它们做出正确的理解。这些例子不一定都要在课堂上给学生讲授，完全可以留给他们课下自己去阅读和推敲。表1列出的是各章例题数量的分布情况。

表1　　本书各章例题分布

章	例题数量	章	例题数量	章	例题数量
1	9	4	17	7	11
2	8	5	19	8	19
3	8	6	20	9	19

3. 对全书中的算法，尽可能地从“算法描述”、“算法分析”和“算法讨论”3个方面进行全方位的讲述：“算法描述”就是用类C语言书写算法，书写中并不特别强调严格的变量说明，以便给读者提供较为宽松的阅读和理解环境；“算法分析”就是对算法的结构、功能做出解释，细说各部分所要完成的任务，指出特殊变量在

算法中的意义和作用；“算法讨论”就是讲述有关算法的改进、扩展、注意事项等。总之，希望通过这3个方面的介绍，能够使读者更好地理解算法的整体功能，更快地掌握算法的实现思想，更多地领悟程序设计过程中应该注意的问题和特殊的技能、技巧。

4. 书中每章的后面，都附有大量的习题，分为4种类型：填空、选择、问答、应用，可供教师备课及学生检验学习效果使用。表2列出的是各章习题数量的分布情况。

表2　　本书各章习题分布

章	习题数量	章	习题数量	章	习题数量
1	34	4	47	7	44
2	45	5	55	8	48
3	42	6	43	9	50

通过人民邮电出版社教学服务与资源网（http://www.ptpedu.com.cn），可以免费下载关于本书习题的参考答案。

5. 本书在出版的同时提供与其配套的PPT课件，它既完整又详尽，能很好地减轻授课教师的备课负担，便于在教学实施过程中使用。

本书的出版，是基于作者多年的教学经验的总结，更是参与写作的每一位同仁齐心协力的成果。在此，对所有为本书顺利出版做过工作的人，特别是沈寄云、梁发寅、陈吉人、蒋玮、王晓宇、宗涛等人，表示诚挚的谢意！

由于编者水平所限，书中肯定会存在错误和不足之处，恳请读者批评指正，更期盼使用本书的教师能够不吝赐教。

编　者

2010年4月

目 录

第 1 章 数据结构概述

“数据结构”是计算机专业的一门重要基础课程，它研究的问题是从实际需要中抽象出来的，是计算机科学各领域以及系统软件都会用到的知识。比如，语言编译程序的实现过程，需要用到堆栈、散列表、语法树；操作系统要使用队列、存储分配表、目录树，来对整个计算机系统的软、硬件资源（如 CPU、存储器、外部设备、文件）实施管理；数据库管理系统工作时，要通过线性表、索引树等对数据进行快速搜索查找；在网络设计技术中，会涉及求解最小生成树、最短路径的问题。这里列举出的内容，如线性表、堆栈、队列、链表、树、图等，都是后面具体章节里所要学习的内容。

本章是全书的基础，将讲述有关数据结构的一些基本概念，主要介绍以下几个方面的内容：

- 数据的逻辑结构；
- 数据的存储结构；
- 算法描述与算法分析技术。

1.1 数据的逻辑结构

计算机是对数据进行处理的工具。具体地说，就是对于一组输入的数据，通过计算机的加工处理，得到相应的输出数据。因此，无论人们要让计算机做什么样的大事、小事、繁杂事、简易事，在用计算机语言编写出程序、执行程序得到处理结果之前，都必须涉及如下的 3 个问题：

第一，确定所要加工处理的数据之间的关系，以便对其进行加工处理时，能够知道一个数据的前面是哪个数据，后面是哪个数据，这种数据间的邻接关系，就是所谓的数据的逻辑结构问题；

第二，确定要对数据做哪些加工处理，是插入、是删除、是查找、还是排序，等等，这就是所谓的算法描述问题；

第三，确定以何种方式把数据存放到计算机的内存，并反映出它们之间的邻接关系，从而有利于对它们进行加工处理，这就是所谓的数据的存储结构问题。

本节首先介绍在现实生活中，很容易总结抽象出的各种数据的逻辑结构，也就是数据间的邻接关系。

1.1.1 数据及数据间的邻接关系

数据是大家非常熟悉的一个词汇，它是信息的载体，是人们用符号来表示客观事物的一种集合。早先，只要提及数据，人们就认为是指能够参加运算的那些数字（比如大家熟悉的整数和实数）。自 20 世纪 40 年代中期计算机问世之后，数据的内涵得到了逐步扩展，计算机加工处理的数据由传统的数字扩大到了字符串（如英文或中文文本）、逻辑值（“真”和“假”）等。随着计算机科学的进一步发展以及计算机应用的深入与普及，当前计算机能够处理的数据更是扩大到了表格、图形、图像、色彩、语音等。

因此，现在把数据（Data）定义为：所有能够输入到计算机中被计算机加工、处理的符号的集合。这就是说，现在说到的数据，已经完全不同于早先人们头脑中理解的“数”的概念了，这是学习本课程必须首先要明确的事情。

数据有各自的类型。可以把计算机处理的数据，笼统地分成数值型和非数值型两大类，它们分别用于数值计算问题和非数值计算问题。当前，计算机大量地被用于文字、表格、声音、图像等非数值计算领域，并对这样的一些数据进行着各种各样的加工处理。

一般地，数值计算问题的特点是加工时的数据量小，处理方法复杂；非数值计算问题的特点是加工时的数据量很大，所处理的数据之间存在着某种特定的关系，计算机在处理它们时，需要对那些关系进行表示。至于具体的加工处理，其方法相对简单得多，不再局限于单纯的数值计算，而是要对它们进行组织、管理、维护、检索等。

通常，数据是由一个个数据元素（Data Element）（简称元素）集合而成的。在本书的不同场合，数据元素也常被称作结点、顶点、记录。每个数据元素都具有完整、确定的实际意义，是数据加工处理的对象。比如，表 1-1 列出的是某公司雇员的信息，每个雇员的信息就是一个元素，由它反映各雇员的实际情况。

表 1-1　　某公司雇员的信息

雇员号	姓名	年龄	性别	住址
071253	庄严肃	34	男	滨江东路蔷薇新村 66 号
073825	刘歆子	28	女	春林路北街 34 号
072154	陈希平	24	男	柳浪街松林坡 12 栋 8 号
071546	李先鸣	30	男	东春斜街甲 77 号
076671	钟永恬	35	女	锦思门前街 2 门 68 号
074533	黄晋民	28	男	解放大桥街 66 号

一个数据元素可细分成由若干个数据项（Data Item）组成，数据项也常称作字段、域。数据项是数据元素中不可再分割的最小标识单位，通常不具备完整、确定的实际意义，只是反映数据元素某一方面的属性。如雇员信息中的雇员号、姓名等，都是一个数据项（或一个字段、一个域）。

大千世界里，每一个数据的实际意义千变万化。在数据结构中，人们并不关心数据元素的实际意义，只是抽象地把它们视为是被加工的一个个对象。数据结构所关心的，是从一个数据能够找到另一个数据的那种“关系”，人们需要根据那种关系来组织和存储数据，这样才能顺利地查找到它们，才能有效地实现对数据的各种处理（如插入、删除、更新等）。

如果两个数据结点之间有着某种逻辑上的联系，那么就称这两个结点是邻接的。若用圆圈代表结点，用结点间的一条连线代表它们之间存在的逻辑关系，那么，就可以用图 1-1 来表示结点 A 和 B 是邻接的。在现实世界里，数据间的逻辑关系，可以具体体现为是前后关系、上下级关系、父子关系、连接关系等。

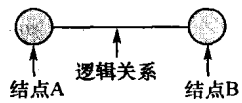

图 1-1　结点的邻接

1.1.2　数据的逻辑结构

从大量实践中抽取出来的、常见的数据间的邻接关系有 3 种：线性关系、树形关系以及图状关系。数据间的邻接关系，就是数据的逻辑结构（Logical Structure），它只体现数据元素间的抽象化关系，不涉及数据元素在计算机中具体的存储方式。因此，数据的逻辑结构是独立于计算机的。

1. 线性关系

所谓数据间具有线性关系，是指数据一个接一个地排列成一行。把数据组织成这种线性关系，可能是人们最常见、最熟悉的做法了。上面所给出的公司雇员表，其数据元素之间就是一种线性关系。如果所要处理的数据之间呈线性关系，那么就说它的逻辑结构是线性的。

在线性关系中，把排在第 1 个位置的结点称为起始结点，把排在最后一个位置的结点称为终端结点，其余的结点都称为中间结点，如图 1-2 所示。

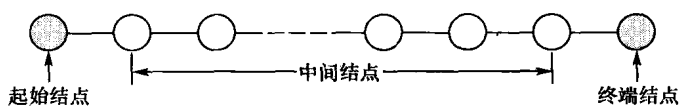

图 1-2　线性关系中的各种结点

数据间线性关系的特点是：除起始结点和终端结点外，每个结点的前面有且只有一个结点与它相邻接，每个结点的后面也有且只有一个结点与它相邻接，起始结点的前面没有相邻接的结点，终端结点的后面没有相邻接的结点。简单地说，数据间线性关系的特点就是：有头有尾，顺序排列。

2. 树形关系

所谓数据间具有树形关系，是指在数据之间具有分支、层次的逻辑关系。由于这种逻辑关系看上去很像自然界的一棵倒置的树：树根位于最上面，树的叶子在最下面，故而得名为树形关系。于是，如果所要处理的数据之间呈树形关系，那么就说它的逻辑结构是树形的。

图 1-3 所示是一个树形结构图例。人们所熟悉的文件目录间的逻辑结构就是树形的。

数据结点间这种分支、层次式组织形式的特点是：第 1 层只有一个结点，它是树形关系的起点；除第 1 层结点和分支末端结点外，位于中间各层结点的前面只有一个结点与它相邻接，每个结点的后面可以有多个结点与它相邻接；第 1 层结点的前面没有结点与之邻接，每个分支末端结点的后面没有结点与之邻接。

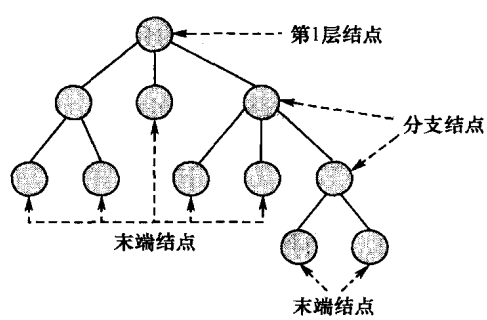

图 1-3　树形结构图例

3. 图状关系

如果数据中的任何两个元素之间都可能有邻接关系，那么就说它们之间的关系是“图状

的”。于是，如果所要处理的数据之间呈图状关系，那么就说它的逻辑结构是图状的。不难理解，图状关系是数据间最复杂的关系。

交通网络或通信网络都是图状关系的典型例子。比如，图 1-4 所示为一张航空网络图，图中与结点“武汉”相邻接的结点有“北京”、“广州”、“南京”，而与结点“南京”相邻接的结点有“北京”、“广州”、“武汉”、“上海”。在图状关系中，找不到谁是起点，谁是终点，各个结点的地位可以说都是相同的。

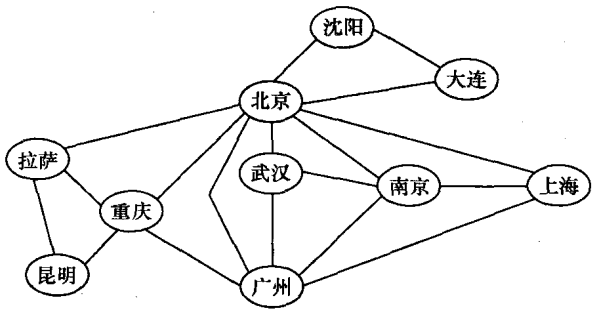

图 1-4　航空网络

图状关系的特点是：每个结点都可能与多个结点有邻接关系。数据间的线性关系和树形关系，都可以视为是图状关系的一个特例。

1.1.3　数据逻辑结构的形式化描述

人们在表示两个事物间存在固定关系时，常采用的方法是使用“偶对”。偶对分无序偶对和有序偶对两种。如果两个事物间的关系没有次序之分，那么这时的偶对为无序的；如果两个事物间的关系是有次序的，那么这时的偶对为有序的。

为了表示无序偶对，用圆括号把两个对象括起来，对象间用逗号隔开；为了表示有序偶对，则用尖括号把两个对象括起来，对象间用逗号隔开。比如，a 和 b 是邻居，那么 b 和 a 也一定是邻居，因此“邻居”这种关系是无序的，所以既可以用(a,b)，也可以用(b,a)来表示 a 和 b 之间的邻居关系。又比如，a 是 b 的上级，那么 b 就是 a 的下级，因此“上下级”这种关系是有序的，所以只能用<a,b>，而不能用<b,a>来表示 a 和 b 之间的上下级关系。

有序偶对表示的关系是有次序的，常称偶对的第 1 个元素是第 2 个元素的“直接前驱”，简称“前驱”；称偶对的第 2 个元素是第 1 个元素的“直接后继”，简称“后继”。比如<a,b>中 a 是 b 的前驱，b 是 a 的后继。

数据的逻辑结构 L 反映出数据间存在的逻辑关系，可以借助偶对通过一个二元组来表示数据间的各种逻辑关系。比如，K 是数据元素的有限集合，R 是集合 K 到集合 K 的一种关系的集合。那么数据间的逻辑结构 L 可表示为：

$$L=(K, R)$$

例如，a 有两个儿子 b、c；b 有一个儿子 d；c 有三个儿子 e、f、g；e 有两个儿子 h、i。那么可以用二元组 F=(K, R)来表示他们之间的“父子”关系，其中：

K={a, b, c, d, e, f, g, h, i}；

R={<a, b>, <a, c>, <b, d>, <c, e>, <c, f>, <c, g>, <e, h>, <e, i>}。

一般地，若 $k_i \in K$，$k_j \in K$，$<k_i, k_j> \in R$，则称 k_i 是 k_j 相对于关系 R 的直接前驱（简称

前驱）；k_j 是 k_i 的直接后继（简称后继）。如果 K 中的某个结点没有前驱，则称该结点为起始结点；如果某个结点没有后继，则称该结点为终端结点；K 中除起始结点和终端结点外的其他结点统称为内部结点。

对于一个逻辑结构 L=(K, R)，如果它只有一个起始结点和一个终端结点，其他的每个结点有且仅有一个前驱和一个后继，则称 L 是线性结构；如果它有一个起始结点和多个终端结点，除起始结点外，每个结点有且仅有一个前驱，则称 L 是树形结构；如果每个结点都可以有多个前驱和后继，则称 L 是图形结构。树形结构和图形结构都是非线性的结构。

例 1-1　某车间有师傅 T，工人 W_1、W_2、W_3 是 T 的徒弟，工人 G_1、G_2 是 W_1 的徒弟，工人 G_3、G_4 是 W_2 的徒弟，工人 G_5 是 W_3 的徒弟。试用二元组表示他们之间的师徒关系，画出结构图。请问这是一种什么结构的关系？

解：所给师徒关系的二元组可以表示如下。

$$L=(K, R)$$

其中：

$K=\{T, W_1, W_2, W_3, G_1, G_2, G_3, G_4, G_5\}$

$R=\{<T, W_1>, <T, W_2>, <T, W_3>, <W_1, G_1>, <W_1, G_2>, <W_2, G_3>, <W_2, G_4>, <W_3, G_5>\}$

这些数据之间存在的是一种树形关系，结构图如图 1-5 所示。

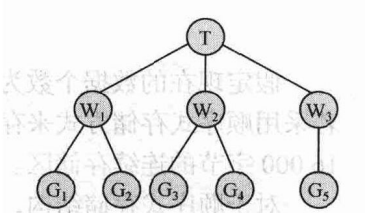

图 1-5　师徒间的树形关系图

1.2　数据的存储结构

无论数据间有何种逻辑关系，它要得到计算机的加工处理，就必须存放到内存中才能进行。注意，这时存放数据并不是一个简单的问题，既要存储数据本身，也要存储数据间的邻接关系。因为只有这样，在对数据进行加工处理时，才能够方便、正确地从一个数据找到与之邻接的另一个数据。

数据的存储结构（Storage Structure），就是研究数据在内存中的存储方式，也就是在内存中有哪些存放数据的方法。数据的存储结构在有些书里也称为数据的物理结构（Physical Structure）。从整体上看，数据在存储器内有两种存放的方式：一种是集中地存放在内存中的一个连续的存储区；另一种是利用存储器中的零星区域，分散地存放在内存的各个地方。

在把数据存储到存储器时，是以数据元素（即数据结点）为单位进行的。分配给一个数据结点的存储区域，称为一个存储结点。如前所述，数据存储在存储器里，既要存储数据本身，还要存储数据间的邻接关系，因此，在一个存储结点里，除了要有数据本身的内容外，还要有体现或反映数据间邻接关系的内容。

1.2.1　顺序式存储结构

所谓数据的顺序式存储结构，即是为一组数据分配一个连续的存储区，然后按照数据间的邻接关系，相继存放每个数据。因此，使用这种存储结构，在每个存储结点里只需存放数据元素本身，不需要有反映数据邻接关系的显式信息。数据的这种存储结构，是借助存储结点间的位置关系，来体现数据元素之间的邻接关系的。

比如，图 1-6 左侧所示为一个数据元素所需要的存储尺寸，本书约定将这个尺寸记为 size

字节。图 1-6 右侧所示为在内存里开辟了一个连续的存储区，用来依次存放数据的若干个存储结点。

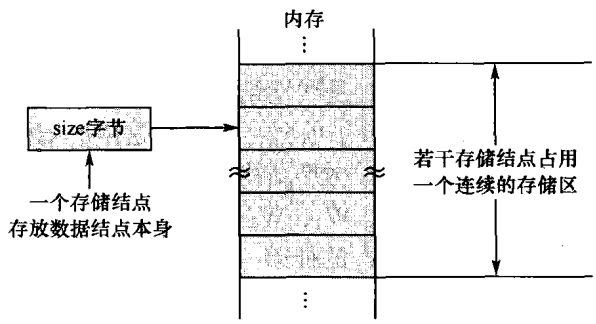

图 1-6　顺序存储结构

假定现在的数据个数为 1 000，每个数据元素需要占用 size =16 个字节的存储区。那么，若采用顺序式存储方式来存放这 1 000 个数据，就需在内存里为它们开辟一个有 1 000×size= 16 000 字节的连续存储区。

对于顺序式存储结构，要再次强调的是在每一个存储结点里只存放数据，不存放数据间的邻接关系，数据间的邻接关系是借助存储结点本身一个紧接一个存放的位置关系体现出来的。因此，顺序式存储结构对存储的利用率为 100%（即分配给存储结点使用的存储区全部被用来存放数据内容）。可以看出，邻接关系为线性的数据，使用顺序式存储结构最为合适，因为它的存储利用率最高。

1.2.2　链式存储结构

所谓数据的链式存储结构，即是存储每个数据的存储结点都由两个部分组成，一部分用来存放数据元素本身的信息，另一部分用来存放与本数据元素邻接的数据元素存储结点的位置，即存储指向与之邻接的存储结点的指针（起始地址），通过这些指针反映出数据间的逻辑关系。

比如，图 1-7（a）所示为一个链式存储结点，里面除了存放数据元素（用 Data 表示）外，还存放着一个指针（用 Next 表示）。如图 1-7（b）所示的 3 个数据元素，分别是数据元素 A、数据元素 B、数据元素 C，它们之间的逻辑关系是：数据元素 A 与数据元素 B 邻接，数据元素 B 与数据元素 C 邻接。采用链式存储方式时，存放数据元素 A 的存储结点里，存放着指向数据元素 B 的指针；存放数据元素 B 的存储结点里，存放着指向数据元素 C 的指针；存放数据元素 C 的存储结点里，存放着一个空指针符“∧”，以表示数据邻接关系的结束。

在链式存储结构里，通过一个结点的 Next 指针，可以找到它后面的那个结点（即后继）在内存中的位置。因此，必须另设一个指针，指向该存储结构的第 1 个存储结点。在图 1-7（b）里，用 head 表示这个指针。这样，由 head 就可以找到第 1 个存储结点；由第 1 个结点的 Next 可以找到第 2 个存储结点；……；直到遇见空指针符“∧”时，表示结束。为了简明起见，常把图 1-7（b）表示成图 1-7（c），用符号“→”表示指向下一个存储结点。

由于链式存储结构是通过指针来体现数据元素之间的逻辑关系的，因此，可以用存储区里的零星小区域来存储数据，而不必去动用存储器中的那些大的连续存储区（大的连续存储区可以分配给必须占用连续存储区的地方使用）。从这个意义上来说，链式存储方式提高了存

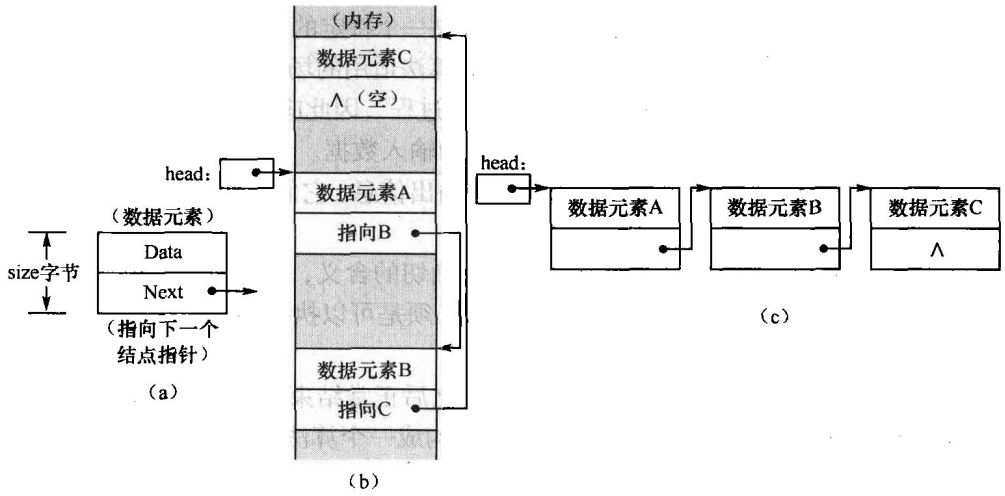

图 1-7　链式存储结构

储器的利用率。另一方面，链式存储结构中的存储结点不仅要存放数据元素，还要开辟适量的存储区来存放指针，以反映数据间的链接关系，这相对于数据内容来说是额外的存储开销。因此从这个意义上说，链式存储结构又降低了存储器的利用率。

比如仍假定现在的数据个数为 1 000，存储每个数据元素需要占用 16 个字节的存储区，存储每个指针需要占用 2 个字节的存储区。那么，一个存储结点就需要占用 size =18 个字节（这 18 个字节当然必须是一个连续的存储区）。这样，如果采用链式存储结构来存放这 1 000 个数据，它们就需要在内存里总共占用 1 000×size =18 000 字节的存储区才行。这就是说，采用链式存储结构时，要比采用顺序式存储方式多耗费 2 000 个字节。

在链式存储结构中，存储结点里的指针并不局限于一个，而是可以根据问题的需要安排多个。如果采用链式存储结构时，存储结点里只有一个指针，则称是单链式结构；如果存储结点里有两个指针，则称是双链式结构；如此等等。这些内容，在后续章节会做详细的介绍。

1.3　算法及算法分析

人们关注数据的逻辑结构（即邻接关系）以及数据在内存中的存储结构，最终目的是要使用计算机对数据进行各种加工处理，比如插入、删除、查找、修改、排序等。

为了实现对数据的加工处理，如果问题很简单，就可以直接用某种计算机程序设计语言编写程序，运行后得出结果。但如果问题较大、较复杂，那么这一过程最好分两步完成：先是通过分析列出与加工处理相关的各个步骤，然后再去用某种计算机程序设计语言编写出相应的程序在计算机上运行。这第一步就是所谓的算法描述，第二步就是所谓的程序实现。

1.3.1　算法及算法的描述

1. 算法和程序的区别

人们常把算法和计算机程序等同起来看待，其实它们是两个不同的概念。

所谓算法（Algorithm），是指解决问题的一种方法步骤或者一个过程。对于一个问题，可

以用多种算法来解决；而一个给定的算法，则只解决一个特定的问题。比如后面要介绍的数据排序问题，就可以给出很多种算法。当然，每一种算法适用的场合以及性能指标是不一样的。

由于算法是“解决问题的一种方法步骤或者一个过程”，因此应该具有以下几个重要的特征。

① 输入：一个算法应该有 n（$n \geq 0$）个初始的输入数据。

② 输出：一个算法可以没有或有一个或多个输出信息，它们与输入数据之间会有着某种特定的关系。

③ 确定性：算法中的每一个步骤都必须具有确切的含义，不能有二义性。

④ 可行性：算法中描述的每一个操作步骤都必须是可以执行的，也就是说，都可以通过计算机实现。

⑤ 有穷性：一个算法必须在经历有限个步骤之后正常结束，不能形成死循环。

例 1-2 判断下面用文字描述的计数过程是否构成一个算法。

① 开始；

② n=0；　　/* 变量 n 赋初值 0 */

③ n=n+1；　　/* 变量 n 增 1 */

④ 重复执行（3）；　　/* 循环执行增 1 操作 */

⑤ 结束。

解： 初看起来，这个计数过程只有 5 个步骤，具有有穷性。但实际上，该过程只要执行起来，就会永远无休止地在变量 n 上面重复做加 1 的操作，形成一个死循环。所以，它并不是一个正确的算法。

例 1-3 编写一个算法，按照从小到大的顺序排列两个数值变量 x、y 的内容，即要求最终有 $x \leq y$。

解： 用文字描述解决这个问题的算法如下。

① 输入变量 x、y 的数值；

② 把两个数值中的小者存放到 x 里；

③ 把两个数值中的大者存放到 y 里；

④ 输出 x、y 的值。

可以看出，上面的描述符合算法的 5 个特征。

所谓“程序（Program）”，是指使用某种计算机程序设计语言对一个算法的具体实现。比如，例 1-3 给出的“按照从小到大的顺序排列两个数值变量 x、y 的内容”的算法，可以用如下的 C 语言程序来实现。

```
#include "stdio"
main()
{
  int x, y, temp;
  scanf ("%d%d", &x, &y);                    /* 从键盘输入两个整型数据 */
  if (x>y)                                   /* 对数据进行比较 */
     {temp = x ; x = y ; y = temp ; }
printf ("x = %d, y = %d\n", x, y);           /* 打印输出 */
}
```

对比算法和程序，可以看出算法侧重于对解决问题的方法描述，即要做些什么；而程序是算法在计算机程序设计语言中的实现，即具体要怎样去做。比如，例 1-3 用文字描述解决

这个问题的算法中，只是讲“把两个数值中的小者存放到 x 里，把两个数值中的大者存放到 y 里”，并不去管到底怎样去进行比较和存放。但是，在用 C 语言程序实现中，则要给出具体的比较和存放方法：

```
if (x>y)
   {temp = x ; x = y ; y = temp ; }
```

可见，严格地讲算法与程序是两个不同的概念。

当然，也可以直接把计算机程序看作是对解决问题方法的一种描述，那么算法和程序就是一回事了。本书为了简化表述的过程，加之学习“数据结构”课程之前都已学习过 C 语言程序设计，因此不去过分地强调“算法”和“程序”的区别。

2. 算法的描述

算法是可以用不同方法来描述的，下面给出几种常见的方法。

算法描述方法 1：使用人们习惯的自然语言来描述算法。

上面的例 1-3 就是用自然语言描述的一个算法。下面的例 1-4 采用的也是这种方法。

例 1-4　用自然语言描述输出整数 1、2、3、…、9、10 的过程。

解：用自然语言描述输出整数 1、2、3、…、9、10 的过程的算法如下。

① 开始；

② 将初始值 1 赋予变量 i；

③ 如果 $i>10$，则转向（7）；

④ 输出 i 的值；

⑤ 将 i 的值加 1，再赋予 i；

⑥ 转向执行（3）；

⑦ 结束。

算法描述方法 2：使用人们熟悉的流程图（即框图）来描述算法。

所谓流程图，即是利用不同形状的图形以及一些带箭头的线条，来描述算法中的各个操作步骤。图 1-8 给出了流程图中可能出现的各种图形的名称和作用。

例 1-5　用流程图描述输出整数 1、2、3、…、9、10 的过程。

解：用流程图描述输出整数 1、2、3、…、9、10 的过程的算法如图 1-9 所示。图中的两个圆角矩形框分别表示算法的开始和结束；两个矩形框分别表示要做的操作；一个菱形框表示条件判断；一个平行四边形框表示输出。

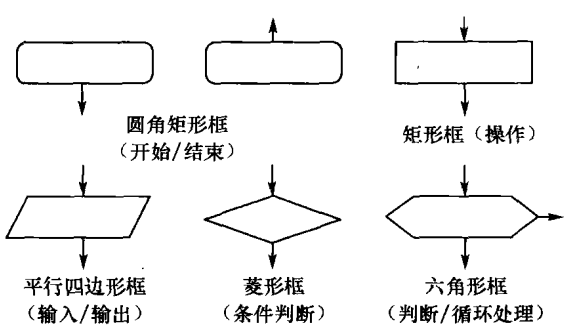

图 1-8　流程图的各种图形名称和作用

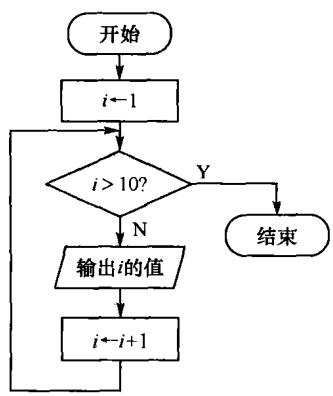

图 1-9　用流程图描述算法