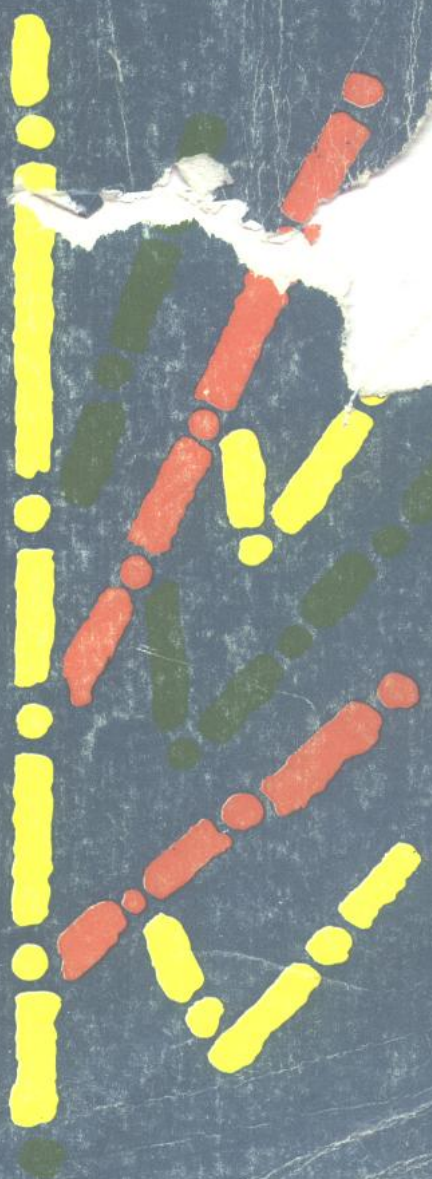
SHIYONGSHUJUJIEGOU

实用数据结构

著者：（美）MILTON ROSENSTEIN PH·D
译者：仝兆岐　叶飞跃　刘兴坤

什么是数据结构
二叉树　N维数组
结构和表示方法
数据库的实现　图的结构
与任务相匹配的数据结构
用BASIC实现N维数组
N维数组的应用
样本数据库的结构
多分支树的基本特性和应用
代数表示法的转换
路径矩阵　数字滤波器
连接计算　链接表　调度程序
中序排序程序　模式匹配
流程结构　遍历转换
有序二叉树的操作



石油大学出版社
SHIYOUDAXUECHUBANSHE

实用数据结构

[美]米尔顿·罗森斯坦 著

仝兆岐 叶飞跃 刘兴坤 译

石油大学出版社

内 容 提 要

本书深入浅出地全面介绍了数据结构中的数组、树、图、队列、表及堆栈等基本概念。在每部分内容介绍后立即给出应用实例是本书的主要特点，而且每个实例都用BASIC和PASCAL两种语言编写。这对于学习数据结构的读者来说一方面可加深对基本概念的理解，另一方面也有重要的参考价值。浅显易懂、实用性强、适用范围广是本书的优点。本书可作为大专院校教材，而对于从事程序设计的技术人员和计算机专业爱好者来说，本书具有重要的参考价值。

实用数据结构

[美]米尔顿·罗森斯坦 著

仝兆岐 叶飞跃 刘兴坤 译

*

石油大学出版社出版

山东省 东营市

东营新华印刷厂印刷

全国新华书店发行

*

开本 787×1092 1/16 11.5印张 280千字

1991年1月第1版 1991年1月第1次印刷

印数：1—2000册

ISBN 7-5636-0125-2/TP·05

定价：3.10元

前　言

该教科书是五年来对大学生们讲授数据结构的成果。起初，学生们都曾认为数据结构课没有多大用处。那时候这门课被严格限制为研究生的课程，因此数据结构讲完了以后再没有别的后续课程来运用所学的这些知识。各种数据结构通常在非结构化的程序课或算法课中提及并加以应用，这远远超出了学生的理解能力和程序设计能力。因此给学生讲授数据结构就好象是用练习来为难学生而不是使学生增长知识。

我想对于教师也存在着同样的问题。不管我如何努力去把那些大如磐石的程序拆成小块；不管我如何试图小心地在课堂上讲解它们，权衡所讲的每一句话，观察每一张面孔是否有迷惑不解、失望或理解的表示；不管我如何难得成功；只要我讲课时有一点点失误，有一瞬间的模糊不清，就足以推翻我常常摇摆的教学方法的"大厦"，并使我的听众陷入歧途。

实际上，学生们是正确的。数据结构课程当时主要是为编译程序课作准备的。只有以编译程序理论为主修目标的学生才会最终发现堆栈、队列、表、数组、树、格等的实际应用价值。而这些概念却都曾使他们头疼过。

同时，程序设计的发展没有受到编译程序理论的影响。程序设计仍然处在一种创作艺术的地位。程序是非结构化的。单纯的算法都是由于偶然的发现而产生的。那些不想编写编译程序，而只想编写一些有用的、面向任务的程序的人们，很明显没有从数据结构和程序工程中得到益处。对此，大多数的学生也深有同感。

数年之后，我们教师才开始懂得什么是应该向学生讲授的东西。在创造新的算法，在对结构化程序的不断要求中，数据结构扮演了极其重要的角色。数据结构现在已被认为是创造算法的最基本的材料。数据结构课程也被看成是算法的基础课程。没有数据结构的基础而想创建算法，就好比没有任何数学知识而想设计电子电路一样。同样，结构化程序设计也显示了它的优越地位。正是由于非结构化程序设计才产生了 Frankenstein 式的怪物，难以理解、交流、调试和修改。现在每天各种软件的崩溃再次证明，庞大的、磐石般的程序再也不能被人们接受了。

对数据结构和结构化程序设计在面向任务的程序设计中所起作用的不断理解和认识，促使我编写了这本书。它至少反映了五年来我和我的学生在教和学两个方面所作的巨大努力。

我力图把教材编写成既面向实际的需要，又能保持理论的严谨。书中的各种数据结构都涉及实际的应用。在描述每个数据结构时尽量使用最简洁的语言，去掉了不必要的复杂部分（也许这部分很吸引人），目前就是要把这些数据结构应用到程序设计中去，而不是将其掩盖在复杂性的背后。因此，书中只要提到一种数据结构就立即把它用到各种有用的实际应用上。在教材的某些地方留给读者一些不完整的程序和过程，读者必须经过一番努力才能认识到程序原来的面目。自然也有一些完整和详细的应用程序，读者完全可以直接使用它们。如数据库、词法检查程序、排队和清单程序、同构和次同构检查程序、检查和分类程序、拼法和符号检查程序、用加入新的运算符来扩展语言功能的程序、数字门电路分

程序等等。我为能编写这些算法和程序而感到自豪。它们从根本上令人信服地证明了数据结构的实用价值。

本书是为所有程序设计者、学习程序设计的学生、程序设计爱好者或专业程序员编写的。最适合于计算机专业一个学期的数据结构课程使用，或其它讲授程序设计课的理工学校使用。它也适合于那些愿意学习程序设计、有毅力且思路敏锐的读者使用。为使本书适应如此广泛的读者需要，所有的程序或过程都用 BASIC 和 PASCAL 两种语言编写，并加了注解。能使用上述两种语言中的任一种编写程序是阅读本书的唯一要求。BASIC 是微机用户的主要语言。PASCAL 在大多数大学中使用得较为普遍。我建议两种语言的使用者都学习一下另一种语言(BASIC 或 PASCAL)以减少对两种语言间认识上的差别。在本书的后半部分，我使用了 SBASIC 语言，目的是能尽量消除这种差别。

虽然这本书是由我编写的，但其中还有许多其他人的辛勤劳动。从我的同行、其它课本的作者那里我也学到了不少的东西；学生们敏捷的思维和提问也曾迫使我去澄清许多尚未完全理解的概念；已故的菲利普·凯恩特罗威兹博士和许多其他的作者都曾给我有益的指导；我的两个女儿路易丝和艾米以及我的妻子丽贝卡，她们的信任和热心是这本书得以完成的根本保证。

纽约理工学院

Milton · Rosenstein, Ph. D.

目　录

第一章 导 论

1.1 引言

学习程序设计，一开始都是编制一系列简单的步骤，让计算机完成一定的任务，这一系列步骤就称之为算法，它确定了程序的特征。如果算法很差，会使编写的程序冗长，难以理解、调试和修改。而一个好的算法将有助于你编写出一段简洁的程序。显然，选择算法这一步是相当重要的。

但是，大多数的程序员在选择或编制一个算法时，常会产生一些不正确的想法，这是因为他们往往把算法看成是艺术而不是科学，这些算法大多数是从某本书上抄来或凭空靠直觉而产生的。他们并不知道评价一个算法的标准，而是首先抓住看起来似乎正好满足要求的东西，因此常常把一个不恰当的算法看成是好的。

建立算法与创造艺术不一样，靠的不是直觉。当今发展得已很完备的程序工程，给程序设计提供了相当有价值的帮助，特别是给建立算法提供了强有力的工具。遵照程序工程的方法来做是相当有效的，它使算法的建立更具有科学性。只要你有决心和信心去开发最先进的程序，是很容易掌握这种方法的。

程序工程指出，数据结构是建立算法的基础。而选择什么样的数据结构确是一个关键的决策，它直接影响了程序的开发。选择一个合适的数据结构便很容易形成一个简洁有效的算法；相反，如果选择的数据结构不恰当，则会带来相当大的麻烦，甚至于不能形成一个算法，不能完成我们的任务，更糟糕的是，也许算法能够形成，但最终的程序却是冗长而十分庞大的，让你难以理解、调试和修改。下一节我们将讨论为什么会这样。

1.1.1 什么是数据结构

数据是程序加工的原始材料，它可能是数字、字符串或两者兼而有之；它也许是模－数转换器的输出，即采样了的电子讯号；它也许是从穿孔纸带、磁带或磁盘上读出的一串二进制数；也许是调制－解调器上将电话中的声信号转换成计算机可接收的格式输出；也许是(通常是)键盘操作的输出。

程序的功能一般依据输入数据来确定，任何程序都必须处理数据，把它从最初的格式转换成结果所需要的格式。例如，一个简单的用来求输入数字的和的程序，就是将一串数字(数据)转换成一个和(结果)。更复杂的程序如记帐程序，是将业务往来的数据转换成收入和支出的报表。

用这种方式来定义程序是否有意义呢？我们的回答是肯定的。至少，程序是将原始资料转换成最终格式的一个或一组工具。不管是谁，只要他制造过东西，就会明白工作的困难程度决定于使用什么样的原始材料。如要造一个木桌，桌子的各个部分都是现成的，那当然很容易，只要将它们装配起来就行了；但若使用的木料需要到木材场去取，那就困难

多了；如果能用胶合板来做桌面，那也比将一块块木板拼成桌面要简单得多。因此，在干活之前如不仔细考虑好使用的材料和工作方式，那他就是一个笨木匠。

脑力劳动也是如此。原始材料(数据)的形式或结构也同样影响着任务的难度。例如，考虑一个两数相乘的数学问题，看起来很简单，让我们试试将下面的两数相乘：

$$\text{LXVII} \times \text{CLXIV} = ?$$

你看这并不简单吧，将罗马数字相乘不是件容易的事。但如果数字是以不同的形式给出，比如是阿拉伯数字

$$67 \times 164 = ?$$

这就很简单了。两个问题其实是相同的，代表了同样的值。很显然，任务的困难程度关键决定于原始材料(数据)的形式。

这样的例子可以说是举不胜举。如果你曾经认为任务的难度取决于任务本身，那么现在就该重新考虑一下了。一个任务同时既可能是容易的也可能是困难的，前面的例子就证明了这一点。在这个例子中，任务的难度几乎完全依赖于数据的形式。难度并不是任务特有的特征，但却影响着我们如何去实现这个任务。

这个结论同样也适用于程序设计。程序设计任务并不依据其难度来评价。一个考虑不周的实现方法可能生成一段很长的、难以理解和调试的程序。但另一种能完成同样任务的实现方法，却可能生成一段简洁明了的、容易调试的程序。如果你真正接受了这个概念，那么，对于一个冗长的程序来说，只能证明其中可能存在问题，而并不能说明其任务是十分困难的。

程序处理的数据结构很显然是影响程序开发的最重要因素，也是主要控制程序设计的原始材料。简洁而高效的程序设计自然依赖于数据组织的方式。真正认识到这一事实的程序员，他就能向高水平的程序设计迈出巨大的一步。

用结构来代替组织这个词，就确定了数据结构这个名词。物理结构有一定的形态，它是各个部件以某种位置关系组合而成的。数据结构与之没有什么不同，它也是各个元素以某种关系组合起来的。数据结构这个复数名词代表了一个数据的多种组织方式。

多年来，人们对数据结构进行了大量的研究和分析，大多数据结构都用于满足开发编译程序软件的需要。而这样的大型程序要求使用尽量少的内存空间，运行速度要尽可能快而且没有错误。这时常规的程序设计方法已不能满足要求，因此人们开始使用程序工程中的许多高级技术，其中之一就是数据结构技术。

1.2 简单的数据结构

现在让我们看看一些特殊的数据结构，这样可以从中知道什么是数据结构，它是怎样供程序员使用的。这里我们只作简单的介绍，以后各章将会对其加以详细讨论。

1.2.1 表和多重链表

大家都熟知的表就是一种数据结构。数据元素(数字或字符串)按行或列组成，一般地，行代表一个单独的项；列代表这个项中的一个特征。图 1.1 就是一个表。

表的组织方式自然地确定了对它的操作。我们可以很容易地按库存量一列相加求得现存的库存量。也可以按价格一列相加得到总的价格。如果我们希望找到某个货物的价格，可以按第一列向下寻找，找到这个货物后再按该行平移到价格这一列(如果货物也按

字母顺序排序，这个过程会加快很多）。表结构有许多优点，这就是它得以普及的原因。

货　物	价　格	库　存	需订货数
别　针	$5/罗	200罗	50罗
风纪扣钩	$1/打	300打	50打
风纪扣圈	$3/罗	50罗	50罗
锁	$2/个	500个	100个

图1.1　表

对于表，有些操作是不适用的。如果我们按货物名称排了序，那就不能按价格的高低来排序；同样，当很方便寻找每个货物时，就不很容易找出最贵、最便宜或次贵的货物是什么。当然，再建一个表按价格的大小来排序能够解决这一问题，但这样做又带来了一些麻烦，而且也浪费内存。因此，如果我们希望按照数据项的多个特征来排序，那么表这种结构是不合适的。

这里我们已经开始认识到了正确的数据结构的重要性。

假如我们有许多货物并定义了名称、价格、库存量等等特征，我们需要挑选出某一个货物并打印它的名称、价格、库存量等全部特征，很明显应使用如下算法：将货物按名称的字母顺序排列在表结构中，当给出了货物的名称后，寻找表的第一列，直至找到这个货物，然后打印出该行全部的内容。如果没有找到这个货物，则打印“库存中无此货物”。表很适合于这类任务。这种算法也是直接和明显的。

显然，下一步我们该满足更多的要求（这种要求确实困难多了，但不是任务本身决定的）。除了根据名称来选择货物外，顾客也许会根据价格、库存量、最后订货日期或其他的数据项来选择货物。

如果我们使用先前的那种表，那么问题就来了，算法不可能还是那么简单。也许，我们会按照价格来重新排这个表，然后再根据给定的价格来查找。如果给定的是库存量，那么我们将表重新排序一次。当然按照这个算法来编写程序是可以的，但应该说这是一种笨办法。

另一种数据结构是多重链表（以后再讨论），它能十分有效地解决这个问题。这种数据结构允许程序员建立一种带有所有数据的表。这种结构很类似于对数据的每个特征都排了序。多重链表只需要建立一次，但却可以根据任意一个特征来搜索，最终的程序将会是既简洁又易于理解和调试的，而且它比表结构程序的运行速度要快得多。

1.2.2　图

图是一种很类似于 Tinleer Toy 玩具的数据结构。画在纸上，它是由一些圆圈（结点）和相互连接的线段（联系）组成的。圆圈代表数据项，线段代表数据项之间的关系。如图1.2所示的是车站之间的关系图。站 A 与站 D 相连，站 B 与站 C 相连，站 C 与站 A 相连。

图1.2很明显地给出了解决有关连接这类问题的算法。能否从站 A 到站 C 只经过一

个车站呢?如果不行,那么需要经过哪些车站?能否从站 B 到站 D 而不经过站 A 呢?第一个问题可以检查 A 与 C 之间有没有连接而得到答案,如果不行,那么寻找并记录下从 A 到 C 所经过的连接即可。第二个问题同样可以通过搜索这个图而得到答案(稍后,我们将讨论计算机是怎样帮助这种搜索的)。

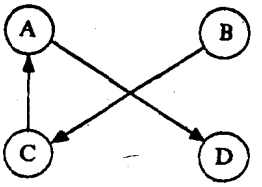


图1.2 车站连接图

假设你接受的任务是编写一个提供火车运行情况的程序,这看起来很复杂,你可能采用的办法是把常见的火车时刻表以表的格式输入到计算机中去,再靠你自己的技巧来编写回答各类询问的程序。当程序完成后,你会发现这个程序是相当长和相当复杂的。但如果你学习了图这种数据结构,就会为它已经提供了解决所有问题的算法而感到惊奇。

1.2.3 二叉树

二叉树同样是由结点和连接线组成的。结点代表数据项,连接线代表结点间的关系。不过二叉树与图并不一样,二叉树类似于一棵实际的树。它只有一个入口点,即根结点,这个结点可以分叉为一个或两个子结点。同样任何一个子结点又可分叉为一个或 两个子结点。图1.3就是一棵二叉树。

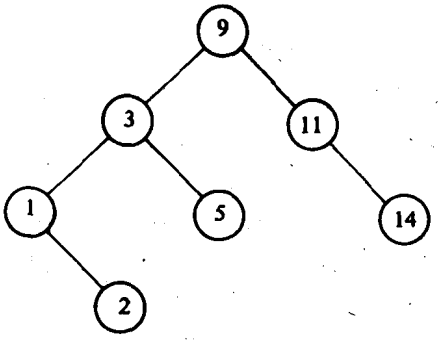


图1.3 二叉树

二叉树具有许多相当重要的应用特性,它是一种有效的搜索结构。寻找树中一个数据

项所花的时间比从表中查找要少得多。如果树的左连接与右连接给定了不同的含义，那么这棵树就可以用于模式识别。例如用于声音和笔迹的识别。二叉树还有其它的一些重要应用，以后我们将会逐渐加以讨论。

1.2.4 其它数据结构

其它的数据结构包括多维数组、递归图、邻接矩阵、单链和双链表、多分支树、篱、索引树、森林和格。这些数据结构并不是全部，因为总是有新的数据结构不断产生。每种数据结构具有独立的作用，它简化了算法并加快了新算法的产生。因此，对这些数据结构的认识和理解有效地提高了程序员创造新算法的能力。再者，这些算法总是既简单又明了，便于产生简洁、好理解、易调试的程序。

1.3 自上而下的程序设计、模块化和数据结构

程序的清晰性可根据调试时所花费的时间来确定。调试所花费的时间越多，程序就越复杂。如果调试的时间很长，就说明这种算法是不确定的，而且不易理解。结果，程序必定是又长又复杂。一个好的程序肯定是又清楚又便于调试。

根据这个要求，我们如何理解一个专业程序员每天只能完成十行的程序设计工作量呢？如何衡量这些专业程序员的最终产品的质量呢？或者相反，如何评价那些作为练习，希望在调试程序上多花费些时间的人所编写的程序质量呢？

公正地讲，每天十行的标准并不适用于短程序，而只适用于相当长的程序。没有一个程序员愿意花费八小时去调试一个只有十行的程序。这个标准只适合于这样的情况：比如一个程序员花了两个星期的时间编写了一个又长又复杂的程序，不得不花费二十个星期的时间去调试它。在这个情况下，这是可以接受并认为是正常的。

程序工程向我们揭示了这样的事实：短程序要比长程序花费少得多的调试时间。那么为什么不可以把一个长的程序分成几个短程序来编写呢？这样的话，每个短程序就可以各自独立调试，然后再连接在一起组成这个长程序。把一个程序分成合适的几段进行编写，它们作为子程序，再用主程序（或称调用程序）把它们组合在一起，这就是模块化的程序。经验证明，这种办法既节省时间又改进了程序的特性。根据这种原则组成的程序叫做结构化程序。图1.4就是一个结构化程序格式的例子。

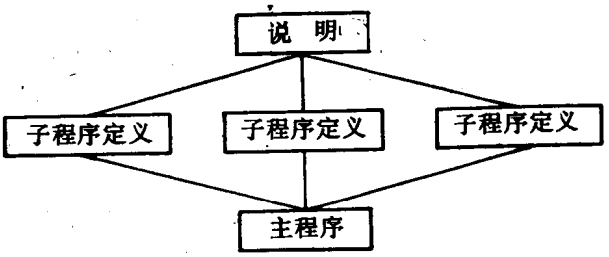


图1.4 结构化程序格式例子

另一种程序工程的方法(自上而下的程序设计)也依赖于模块化技术。自上而下的程序设计是设计和编写程序的一种技术。它从整个工作的全局来考虑尽可能总括地完成程序的各个组成,随后按同样的办法合理地把这一工作分段完成,直至最终编出程序来。这一过程与典型的层次结构完全一致。因此十分适用于大型的程序设计任务,特别是适用于需把任务分派给几个人来完成的场合。当然,对于独立的某个程序员来说,这也是一种相当优异的程序设计方案。这是因为它引导了程序员从任务的起始到完成任务所需努力的几个方面。图1.5是一个典型的自上而下的程序设计方案。

模块化和自上而下的程序设计,都需要把程序设计过程分为几段,但没有一种方法能确保合理地分段。模块化本身并不必然产生高质量的程序。如果分段分得不合理,那么可能产生质量很差的程序。同样自上而下的程序设计方法,也不能保证所计划的各步骤能加速和改善程序设计过程。因为错误的计划将导致整个设计过程的紊乱,这两种方法要想有效地发挥作用,必须依赖于合适、合理地分段。

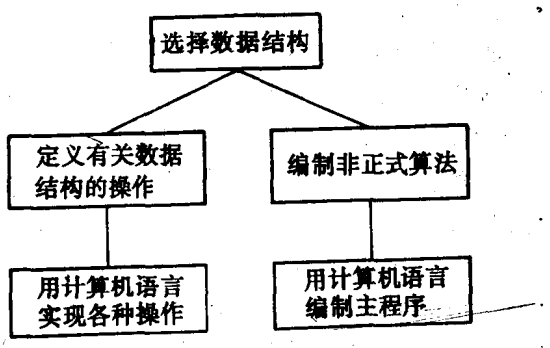


图1.5

数据结构为实现合理地分段提供了有效的工具。不久我们将会看到要选择最优化的算法,必须先确定任务的结构,并使数据结构与它相吻合。数据结构的选择提供了一种好的、第一级的从上到下的实现方法。仅仅使用数据结构的这些操作,程序员就可以编制非正式的算法,并可很方便地测试所选的数据结构是否合适。这种非正式的算法是从上到下设计方法的第二级,并给出了程序的大体概况。最后,用计算机语言去编制这些算法作为子程序,并用主程序将它们连接起来。这一步是最后一级。所有这些过程都为模块化和任务的分段提供了实用的基础(对于使用多种算法的程序,这个过程还需要扩充)。

1.4 这对程序设计者意味着什么?

我们大多数人并不以编制程序为职业,而是为了其它的目的:如科学研究、事务管理或以教育为目的。也许只是简单地作为爱好来开发计算机的应用。不管动机如何,我们的兴趣在于完成高水平的程序,在于尽量节约所花费的时间,在于提高我们编制算法的能力。要做到这一点,掌握程序工程的知识是会有很大帮助的。

习　题

1.1　编写一算法，将三个单词的字以字母顺序排序.描述并分析有关这个作业的思维过程.

1.2　举出一些例子说明一个问题用某种方法去实现时十分困难；但如采用另一种方法却很容易.

1.3　用表结构存放有关货物、价格和库存量的数据。编制一种算法，它允许以最贵、第二贵、第三贵的价格和以库存量来选择任一货物。这种算法是否很复杂？当表的内容相当大时，情况又将如何？

1.4　用数据结构表示如下的飞机航线(画出结构来)：A 市与 D 市，A 市与 F 市，G 市与 A 市，G 市与 D 市，D 市与 A 市，F 市与 G 市。

1.5　假设你领导了一组程序员编写一个程序，程序的功能是在对换支票前检查支票。支票在对换前必须做如下检查：银行有没有这个账号？此账号有没有超支？账号中存款是否足够支付这张支票？建立一个自上而下的设计过程去组织你的程序设计小组。

第二章　N维数组

2.1　结构和表示方法

数组是按某种方式排列的数据项集合。这些数据项可能是数字也可能是字符串，但通常不是两者的混合。元素的排列方式可以是简单的按一维顺序排列，如

1	3	24	12
Joe	Paul	Peter	Sam

或

这些排列通常被称之为向量。排列方式也可以是二维的矩形组合，如：

1	3	5	2
5	2	7	0

(a)

−5	2	0	1	0
0	2	1	3	0
3	−1	2	4	2

(b)

4	3
5	2
3	2
4	3

(c)

Paul	John	Pete
Hal	San	Paul
Ed	Joe	Bill

(d)

Now	is	the	time
for	all	good	men
to	come	to	the
aid	of	their	countey

(e)

这些称之为矩阵(矩阵是平行的)。排列的方式也有三维、四维、五维或更多维。通常将这种多维结构称之为 n 维矩阵，n 可以是任意正整数。

一般，一维数组(A)可由下式表示：

$$A = a_1 a_2 a_3 \cdots\cdots a_k$$

式中"……"表示其它的数据项，a_k 表示最后一项，A 是数组的名，a_1、a_2等是数组的元素，下标1到 k 称为索引(单一索引)。数组 A 由一行 K 列组成。元素所在的列由索引指定。例如 a_5在第五列中。

二维数组一般由下式表示：

$$
A = \begin{matrix}
a_{1,1} & a_{1,2} & a_{1,3} & \cdots & \cdots & \cdots & a_{1,k} \\
a_{2,1} & a_{2,2} & a_{2,3} & \cdots & \cdots & \cdots & a_{2,k} \\
a_{3,1} & a_{3,2} & \cdots & \cdots & \cdots & \cdots & a_{3,k} \\
a_{4,1} & \cdots & \cdots & \cdots & \cdots & \cdots & a_{4,k} \\
\cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\
\cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\
a_{n,1} & a_{n,2} & \cdots & \cdots & \cdots & \cdots & a_{n,k}
\end{matrix}
$$

其中元素 a 的第一个索引表示元素所在的行，第二个索引表示所在的列。例如元素 $a_{3,6}$ 是在第三行第六列的位置。这个数组表示共有 n 行 k 列，它是一个 $n \times k$ 的二维数组。

三维数组也能用空间几何图形表示。它的第一索引表示行，第二索引表示列，第三索引表示在纸面上方的空间位置。更多维数的数组就不可能用简单的几何图形来表示了。

2.1.1 另一种表示方法

这种表示方法不是把索引解释为一组垂直(相互正交)数轴中点的坐标，而是把它们看成是(指向)元素位置的说明。因为我们不可能想象三维几何空间以外的东西，所以不能表示大于三维的数组。但是我们都有如何确定某样东西的经验，而且这往往要三个以上的参数。例如帝国大厦的位置由以下几个数据确定：

国家：	美国
城市：	纽约
区：	迈哈顿
街：	34
号：	125

这里需要五个数据项。如果我们把帝国大厦(ESB)看成是 n 维数组中的一个元素，而索引看成是它的位置，那么这个元素(ESB)就由下式确定：

$$ESB_{美国,纽约,迈哈顿,34,125}$$

且它是五维矩阵中的一个元素。

档案系统给出了一种很好地表示 n 维数组的方法。假设我们有一个只包含单个数据项的档案系统。系统的组织为：数据项所在的行；行所在的页；这一页所放的抽屉；抽屉所处的柜；柜所属的组。每个数据项需要六个值来确定它的位置：组、柜、抽屉、页、行、行中的位置(列)。如果将这个档案系统定义为数组 I，那么，其中任一数据项 i 由 $i_{i,j,k,l,m,n}$ 来确定，式中 i 是组号，j 为柜号，k 为抽屉号，l 为页号，m 为行号，n 为列号。这样 I 是一个六维的矩阵。

2.2 与任务相匹配的数据结构

在本书中我们还要多次强调，最适合于任务的数据结构是与任务结构相匹配的数据结构。档案系统为此做了很好的说明。档案系统中的各数据项要转存到 n 维数组中去是十分方便和迅速的，反过来也是一样。这是两种结构相匹配的最好例子，对数据项相同的操作：一寻找一个数据项(或相邻的一组数据)、搜索或删除这个(或这组)数据、增加一个(或