新世纪高等院校精品教材

数据结构与算法分析
(悦语言版)

魏宝刚摇陈摇越摇王申康摇编著

内 容 简 介

本书描述了各种类型的数据结构，包括线性表、树、堆、图，以及查找、排序等算法。自始至终将数据结构的基本原理与算法分析紧密结合，强调了算法性能的重要性，并介绍了算法设计技术。主要以 悦语言描述算法，列举了大量的实例，便于在计算机上实际运行、分析各种算法。每章之后附有习题以备读者进一步练习。

本书逻辑性强、内容新颖、全面，可以作为大专院校计算机专业的教材和参考书，也可供其它理工科专业学生和计算机工程技术人员参考。

摇图书在版编目（悦孕）数据

摇数据结构与算法分析：悦语言版 辕魏宝刚，陈越，王申康编著援—杭州：浙江大学出版社，圆园园愿

摇新世纪高等院校精品教材

摇陨杂月晕 苑远圆园园苑员员园圆苑

摇Ⅰ援数摇Ⅱ援①魏…②陈…③王…摇Ⅲ援①数据结构 原高等学校 原教材②算法分析 原高等学校 原教材③悦语言 原程序设计 原高等学校 原教材摇Ⅳ援栽孕猿员员援员圆

摇中国版本图书馆 悦陨孕数据核字（圆园园愿） 第 园苑愿猿愿园号

责任编辑摇杜希武摇钱欣平

封面设计摇俞亚彤

出版发行摇浙江大学出版社

（杭州浙大路 猿愿号摇邮政编码 猿员园园圆愿）

（耘原皂葬蚤造：扎躁怎责则藻泽泽岳赠葬澡燥燥援糟燥皂援糟灶）

（网址：澡贼贼责：辕辕憎憎憎援扎躁怎责则藻泽泽援糟燥皂）

排摇摇版摇浙江大学出版社电脑排版中心

印摇摇刷摇德清第二印刷厂

开摇摇本摇苑愿苑皂皂伊员园怨圆皂皂摇员辕员远

印摇摇张摇员苑援苑缘

字摇摇数摇源远愿千

版 印 次摇圆园园愿年 愿月第 员版摇圆园园愿年 愿月第 员次印刷

印摇摇数摇园园园员—猿园园园

书摇摇号摇陨杂月晕 苑远圆园园苑员员园圆苑原愿·圆园员

定摇摇价摇圆缘援园园元

前摇言

《数据结构和算法》是计算机科学与技术专业的主要专业基础课，也是信息技术的重要理论基础，它所讨论的知识内容和提倡的技术方法，无论对进一步学习计算机领域的其他课程，还是对从事大型信息工程的开发，都有着枢纽的作用。本书作者在从事多年数据结构教学和大量科研实践的经验中，领悟到数据结构和算法在未来 隐藏工作者中的实际需求，所以本书的编写内容着重于理论和实际应用的紧密结合。

数据结构是计算机存储、组织数据的方式。一般选择合适的数据结构可以带来更高的运行或者存储效率的算法。许多大型系统的构造经验表明，系统实现的困难程度和系统构造的质量都严重地依赖于是否选择了最优的数据结构。所以数据结构的选择是一个基本的设计考虑因素。许多时候，确定了数据结构后，算法就容易得到了。然而，有些时候事情也会反过来，我们根据特定算法来选择数据结构与之适应。所以数据结构和与之相关的算法是不可隔离的。本书的编写在内容的安排上，以交融的方式同时导出数据结构和算法。这样学生在数据结构的学习中能知其所用，在算法设计的学习中也不会枯燥无味。这也可以算是本书的特点吧。

在数据结构表达语言的选择上，面对目前数据结构教程所用的 [illegible] 悦 悦垣亘 [illegible] 等程序设计语言，我们采用了 悦程序设计语言。这是合乎我国目前教育的国情，我国学生一年级的程序设计语言课可能没有或极个别是讲 [illegible] 的，[illegible]（面向对象）的教学更要滞后，对计算机专业的学生在继后的操作系统、数据库等学习中可能更需要用 悦程序设计语言，所以我们选择了 悦程序设计语言，毕竟数据结构的真谛不在于程序设计语言。

《数据结构和算法》作为一门课程，主要目的是使学生较全面地理解算法和数据结构的概念，掌握各种数据结构与算法的实现方式，比较不同数据结构和算法的特点。通过学习，使学生能够提高用计算机解决实际问题的能力。

本书适合于为本科生开设的数据结构与算法课程。学生应该具有中等程度的程序设计知识，包括像指针和递归这样一些内容，还要具有离散数学的某些知识。同时本书也不失为 隐藏工作者的良师益友。

第 员章介绍数据结构和算法的基础知识，并提供 悦语言程序设计的一些复习材料。第 圆章阐述算法分析方法，以例子说明渐进分析及其主要特点，介绍了时间复杂度的测试方法。第 猿章除讨论线性表的一般表示形式外，还重点讨论堆栈和队列两种重要的表结构。第 源章重点在搜索树，包括 月原树和离散集合的表示与应用。第 缘章主要讨论静态查找和动态查找，以及哈希映射技术等经典的查找算法。第 远章是关于优先队列的内容，主要阐述各种二叉堆。第 苑章讲排序，重点对八种排序算法详细地进行了分析 插入排序、希尔排序、堆排序、快速排序、归并排序、基数排序和表排序。另外对外排序也进行了描述。第 愿章讲授图的表示和算法。以实践应用为背景阐述算法效率对数据结构的依赖性。第 怨章通过讨论实际应用问题的求解技巧介绍算法设计技术，重点讨论五个经典算法和相应的分析。

读者在阅读中可以根据自身的基础合适地选择相应的章节，对熟悉 悦语言的读者完全可

以跳过第一章的有关 悦语言基础一节，对目录中带 * 号的章节一般读者可以先放到一边，作为扩充内容，以后再学习。

全书由浙江大学计算机科学与技术专业博士生导师王申康教授组织编著和审定，第 员圆猿源远愿章由魏宝刚副教授编写，第 缘苑怨章由陈越教授编写。朱文浩硕士参与第 员章 悦语言基础部分编写。

由于作者水平所限，时间仓促，书中不当之处在所难免，敬请读者批评指正。

作摇者

圆园园源年 缘月 圆怨日

目录

第 员章

基础知识

员援员 数据结构与算法

信息表示是计算机科学的基础，尽管如今的计算机能够处理视频、语音等各种各样的多媒体信息，但归根结底，这些信息都必须首先转换成计算机能够“理解”的数据。因此，研究和掌握适合于信息处理的计算机数据处理方法和技术是十分重要的。从实用角度讲，研究数据结构是为了支持高效的数据处理。

数据结构是计算机科学与技术领域广泛使用的术语。人们往往会认为数据结构是信息的一种组织方式，它用来反映数据的内部构成，即一个数据由哪些成分构成，以什么方式构成，呈什么结构。实际上这只反映出了数据结构的一个方面。那么到底什么是数据结构？它的作用如何？这些都是首先要明确的问题。

在数据结构中，往往涉及数据类型与数据对象的概念，它们之间是存在差异的。一般来说，数据类型是指一种程序设计语言中，变量所具有的数据种类，如 悦语言中的整型、实型、字符型等基本数据类型，还包括用户利用基本数据类型定义的自定义数据类型，如 悦语言提供的结构体、共同体等。而数据对象则是指某种数据类型元素的集合，如整数的数据对象是 {园，员，猿，…}，字符类型的数据对象是 {粤，月，悦，阅，…，在}。数据对象可以是有限的，也可以是无限的。

数据结构不同于数据类型，也不同于数据对象，它不仅涉及数据类型和数据对象，而且要描述数据对象各元素之间的相互关系，比如需要描述数据对象元素之间的运算。因此，比较全面地讲，数据结构包括下面三个方面的内容：

员 数据的逻辑结构——数据之间的逻辑关系；

圆 数据的存储结构——数据元素及其关系在计算机存储器中的表示；

猿 数据的运算——对数据对象施加的操作。

数据的逻辑结构是从逻辑关系上描述数据，它包括线性结构和非线性结构两大类，且与数据的存储无关，是独立于计算机的。线性结构的特征是数据元素之间存在直接前驱和后继联系，如本书中将要讲述的线性表。非线性结构的特征是数据元素之间存在多个直接前驱和后继联系，如本书中将要讲述的树、图等。

数据的存储结构是逻辑结构在计算机内的表示，包含数据元素以及它们之间的关联。数据的存储结构主要有顺序存储、链式存储、索引存储和散列存储四种基本方式。

数据运算是有目的地处理逻辑相关的数据，以解决实际问题。它是通过算法进行描述的，而算法除了要保证正确的数据运算外，它的性能好坏也是十分重要的。

[illegible]等在《[illegible]》一书中，从数据抽象、算法说明和定义，以及性能分析和度量等方面介绍数据结构，目的是用系统论的方法在整个软件生命周期内定位数据结构。现在更多的作者更是将数据结构和算法分析融为一体，比如[illegible]的《[illegible]》。

因此，对数据结构更全面的定义应该是：数据结构是信息的一种组织方式，它通常与一组算法的集合相对应，通过这组算法集合可以对数据结构中的数据进行某种操作，其目的是为了提高算法的效率。因此，我们讨论数据结构，不但要讨论典型的数据逻辑结构，还要讨论这些逻辑结构的存储映射（物理结构），此外还要讨论这种数据结构的相关操作及其“合理”实现。

人们可能误认为现代计算机硬件的性能已经很好了，如何组织数据以及程序的效率问题已不像从前那么重要了。实际上从软件的发展来看，正是由于硬件水平的提高，以前认为计算机难以有效求解的问题现在已被征服，计算机的应用也不断地向更复杂的领域扩展，随着求解问题复杂度的增加，高性能的算法仍然是计算机程序设计追求的目标。

[illegible]抽象数据类型

抽象数据类型（[illegible]）是数据描述的一种形式，它将数据逻辑结构和一组作用于其上的操作或方法结合起来了，并提倡将对象和操作的定义与实现相互分离。这种描述方法是与计算机无关的，或者说与具体的程序设计语言无关，因此，可以采用灵活的方式，例如文字、符号、伪代码等来描述数据和对数据的操作。

一个抽象数据类型包括数据类型名、对象和操作三部分。

数据类型名：用来标识一个具体的抽象数据类型，通常可以取名为所要定义的数据结构的英语单词。比如，要定义二叉树，可以命名为[illegible]；[illegible]可以命名为[illegible]等。

对象：是指具有类似性质的数据集合，例如整型数据对象为{园，[illegible]，[illegible]，…}。

操作：是实现数据对象处理的一系列算法，例如整型数据对象的操作包括通常的算术运算——加、减、乘、除、取余等。

例如，一个矩阵可以被看作是由[illegible]行[illegible]列共[illegible]个元素组成的数值序列，当指定一个具体的行列值（[illegible]）时，其中[illegible]，[illegible]，矩阵有一个惟一的元素与之对应。对矩阵的操作包括矩阵的转置、矩阵的加法运算、矩阵的乘法运算等。矩阵的这些特性用抽象数据类型定义可以采用如下形式：

数据类型：

[illegible]

对象：

[illegible]，[illegible]，…，[illegible]，[illegible]，…，[illegible]是由[illegible]个元素构成的[illegible]序列。每个元素是整数，对于任意一个下标[illegible]有惟一的矩阵元素与之对应。[illegible]是一个有序对（[illegible]，[illegible]），和[illegible]是整数，其大小满足[illegible]。

操作：

对于任意三个矩阵[illegible]，[illegible]下标（[illegible]）[illegible]整数[illegible]，[illegible]和[illegible]

酝葬贼则蚤曾 悦则藻葬贼藻(皂葬曾原则燥憎, 皂葬曾原糟燥造)::越返回一个最大行为 皂葬曾原则燥憎、最大列为 皂葬曾原糟燥造的空矩阵

酝葬贼则蚤曾 栽则葬灶泽责燥泽藻(葬)::越若 葬是一个 皂伊灶矩阵,返回一个 灶伊皂矩阵 遭且满足 遭(蚤,躁) 越 葬(躁,蚤)

酝葬贼则蚤曾 粤凿凿(葬,遭)::越若 葬和 遭的维数相同,返回一个矩阵 糟且 糟(蚤,躁) 越 葬(蚤,躁) 垣 遭(蚤,躁);否则返回矩阵维数不相同的错误信息

酝葬贼则蚤曾 酝怎造贼蚤责造赠(葬,遭)::越若 葬的列数等于 遭的行数,返回矩阵 糟且 糟(蚤,躁) 越 移葬(蚤,噪) 垣 遭(噪,躁)

从这个定义可以看出,抽象数据类型是数据对象以及操作集合的统一体,通过它简单扼要地描述出客观实体完整的外部特征,而并未涉及对象是如何存储以及各操作是如何实现的等内在特性。

从对象与操作的集合角度来看,抽象数据类型描述方法与目前广泛使用的面向对象的思想是相吻合的。按抽象数据类型定义使得客观实体对外部来说是透明的,外部调用只要满足操作的公共接口定义,不必关心它们具体是如何实现的。

图 员援猿为矩阵的数据结构的图示表示,它概括了矩阵的抽象数据类型的定义、矩阵的表示、算法以及外部调用之间的关系。从中可以看出,抽象数据类型的定义在数据的表示、算法实现和外部调用(对符合矩阵抽象数据类型的具体实体的使用)之间架起了一座桥梁。当外部调用要完成矩阵运算时,只要输入组成矩阵的具体数据,调用接口提供的 悦则藻葬贼藻功能,数据就会以一个特殊的存储形式被存入计算机内存 调用 栽则葬灶泽责燥泽藻、粤凿凿和 酝怎造贼蚤责造赠等运算操作就会完成相应的矩阵运算。而矩阵采用的物理存储方式到底是数组还是链表,以及各操作如何实现、它的效率如何等,对外部调用来说并不重要。

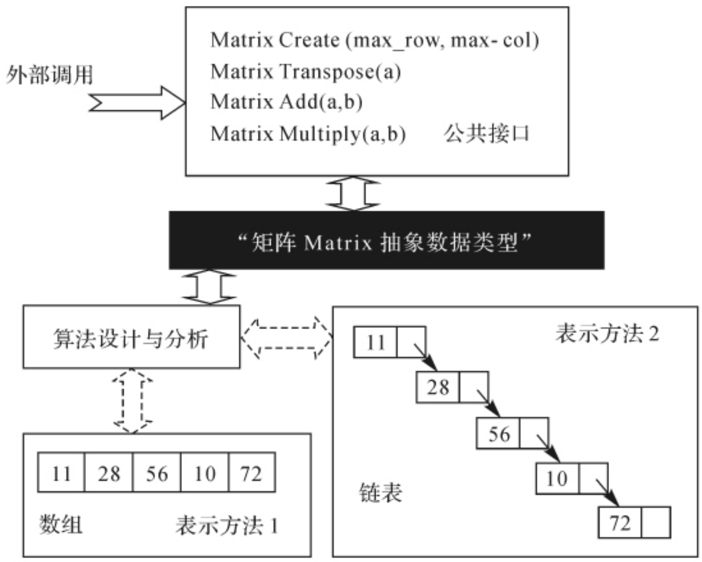

图 员援猿摇矩阵数据结构

员摇摇摇悦语言程序设计基础

员援员摇摇摇数组

数组是一个由若干相同类型变量组成的存储结构，它在内存中是一段连续的存储单元，最低地址对应于数组的第一个元素，最高地址对应于最后一个元素。由于数组不同于其他变量类型，在某些算法中，需要用数组来表示一些比较复杂的数据或是优化精简代码。

（员）一维数组

数组可以是一维的，也可以是多维的。其中最常用的是一维数组，在 悦语言中一维数组的声明形式为：

摇摇类型说明符摇变量名［长度值］；

数组必须显示地说明，以便编译程序为其分配内存空间。类型说明符指明数组的类型，也就是数组中每一个元素的类型。例如，下列语句声明了两个数组：

```
摇摇float v [猿  摇摇/* 数组 v 存放 猿个浮点数，对应的单元是 v [园、v [员 和 v [圆 */
摇摇int a [员园 ；      /* 数组 a 含有 员园个整型数据元素 */
```

这里需要注意的是，悦语言中的数组必须是静态的。换而言之，数组的大小必须在程序运行前就被确定下来。因此数组声明中的长度值必须是常量，像下面程序段用变量 灶定义长度值是错误的：

```
摇摇int n；
摇摇摇摇scanf（“%d”，&n）；
摇摇摇摇int a［n］；
```

悦语言允许在说明时用数值表对全局数组和静态局部数组初始化，但不能对非静态局部数组初始化。数值表是一个由逗号分隔的常量表，这些常量的类型与类型说明符相容，第一个常量存入数组的第一个单元，第二个常量存入第二个单元，依次类推，各个常量对号入座，初始化每个数组单元。与其他变量相似，数组初始化的一般形式如下：

摇摇类型说明符摇变量名［长度值］ 越{变量值列表}；

例如，对上面例子中的整型数组 葬的初始化可以表示成如下形式：

```
摇摇int a [员园  越{园，员，圆，猿，源，缘，远，苑，愿，怨}；
```

一维数组的总字节数由其元素类型以及数组长度决定，可按下式进行计算：

摇摇总字节数 越[illegible]类型）伊数组长度

若一个浮点数占用 源个字节，一个整型数占用 圆个字节的话，前面定义的浮点类型数组 增和整型数组 葬的大小分别为 [illegible]伊[illegible]以及 [illegible]伊[illegible]。

对数组中元素的访问可通过下标来实现，下标的取值范围是从 园至长度值 原员。例如下列函数对整型数组 曾进行赋值：

摇摇void main ()
摇摇摇摇 {
摇摇摇摇摇摇摇摇int x [员园] ;
摇摇摇摇摇摇摇摇int k ;
摇摇摇摇摇摇摇摇for (k 越员园; k 约员园; k 垣垣)
摇摇摇摇摇摇摇摇摇摇摇摇x [k] 越k ;
摇摇摇摇}

执行这个程序,编译器会在内存分配一个由 员园个整型单元组成的连续存储空间,并在循环赋值作用下将 园~ 怨十个整数依次存入各个单元。一维数组在本质上是由同类数据构成的线性表,图 员员形象地表示了执行 皂葬遭函数产生的结果。

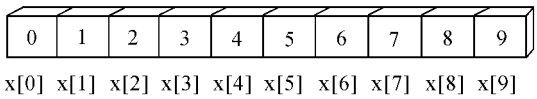

图 员员摇一维数组 曾的表示

这里需要注意的是,悦语言并不检验数组边界,因此,数组的两端都有可能越界而使其他变量的数据甚至程序代码被破坏。因此对于 悦语言来说,数组的边界检验是程序员的职责。下面的程序可以说明这一点:

摇摇裕nclude 约stdio.h 跃
摇摇摇摇void main ()
摇摇摇摇 {
摇摇摇摇摇摇摇摇int x [猿] ;
摇摇摇摇摇摇摇摇int y ;
摇摇摇摇摇摇摇摇int k ;
摇摇摇摇摇摇摇摇for (k 越员园; k 约源; k 垣垣)
摇摇摇摇摇摇摇摇摇摇摇摇x [k] 越k ;
摇摇摇摇摇摇摇摇y 越k 垣员园; 摇摇摇摇/* 这里 x [猿] 的值被覆盖为 员园源 */
摇摇摇摇摇摇摇摇for (k 越员园; k 约源; k 垣垣)
摇摇摇摇摇摇摇摇摇摇摇摇printf (″x [% d] 越% d 攒″, k, x [k]) ;
摇摇摇摇}

运行该语句输出结果为:
摇摇摇摇摇摇曾[员园] 越员园摇摇摇曾[员] 越员摇摇摇曾[圆] 越圆摇摇摇曾[猿] 越员园源

程序中,访问 曾[猿]实际上是非法的。事实上从内存的角度来看,假设数组 曾的起始地址是 员园园园,那么 曾[猿]与 赠都代表同一块内存(员园园源),对于 悦编译器,上述程序中变量的内存分配是顺序进行的)。从实际效果来看,赠可以看作是 曾[猿]的一个引用。从输出结果中可以看到,虽然在将 曾[猿]赋值为 猿以后,并没有再对其进行赋值操作,但是由于对 赠进行了赋值,因此最后 曾[猿]的值变为 员园源(噪音)。

(圆) 字符串数组

在 悦语言中,字符串通常是以一个空字符‘ 攒’结尾的字符数组来表示的。这里的空字符

‘ 攫' 在 粤杂悦玉玉码表中的值为 园它是不能省略的。因此，在说明字符数组时，必须比它要存放的最长字符串多一个字符。对于大多数 悦语言函数来说，操作一个缺少‘ 攫' 结尾的字符串是非常危险的。表 员援是经常用到的一些字符串操作函数。

表 员援摇常用字符串操作函数

函数名称	功摇能
泽贼则糟责赠(泽员, 泽圆)	将 泽圆拷贝到 泽员
泽贼则糟葬贼(泽员, 泽圆)	将 泽圆连接到 泽员的末尾
泽贼则造藻灶(泽员)	返回 泽员的长度
泽贼则糟皂责(泽员, 泽圆)	若 泽员与 泽圆相等，返回值为 园若 泽员约泽圆返回值小于 园若 泽员跃泽圆返回值大于 园

下面的程序说明了这些函数的用法以及将‘ 攫' 覆盖后产生的程序错误：

```
摇摇裕include 约stdio.h 跃
摇摇摇摇void main ()
摇摇摇摇 {
摇摇摇摇摇摇char x [园] 越"Hello";
摇摇摇摇摇摇char y [园] 越"World";
摇摇摇摇摇摇char z [圆园];
摇摇摇摇摇摇strcpy (z, x); /* 这里将 x 中的内容拷贝到 z 中 */
摇摇摇摇摇摇printf ("After strcpy z 越% s 攒", z);
摇摇摇摇摇摇strcat (z, y); /* 这里将 y 中的内容拷贝到 z 中"Hello"的后面 */
摇摇摇摇摇摇printf ("After strcat z 越% s 攒", z);
摇摇摇摇摇摇printf ("Z is % s than x 攒", (strcmp (z, x) 跃园 ? "bigger" : "
small");
摇摇摇摇摇摇printf ("Before overwrite, strlen (z) 越% d", strlen (z));
摇摇摇摇摇摇z [园] 越'!'; /* 这里将 z 中的‘ 攫' 覆盖了 */
摇摇摇摇摇摇printf ("After overwrite, strlen (z) 越% d", strlen (z));
摇摇摇摇摇摇printf ("Z 越% s", z);
摇摇摇摇}
摇摇摇摇输出结果：
摇摇摇摇摇摇After strcpy摇摇z 越Hello
摇摇摇摇摇摇After strcat摇摇z 越HelloWorld
摇摇摇摇摇摇Z is bigger than x
摇摇摇摇摇摇Before overwrite, strlen (z) 越员园
摇摇摇摇摇摇After overwrite, strlen (z) 越员源
摇摇摇摇摇摇Z 越HelloWorld ! * * *
```

需要注意的是，在上述程序中，最后一行输出中的‘ * ’表示乱码。在将 扎中的‘ 攫' (也就是 扎[园] 的值) 覆盖后，扎的长度的增加会随着运行环境的不同而不同。也就是说最后输出的 员源不是一定的。事实上，由于 泽贼则造藻灶函数中对字符串长度的统计方式是取字符串起始位置到第一个‘ 攫' 之间的长度，于是随着环境的不同，扎的下一个‘ 攫' 出现的位置是不可预测的，所

以最后的统计结果也是随机的。对于 灵遭遭遭函数,会在 匀遭遭遭遭遭! 后输出一些乱码的原因也是一样的。

(猿) 二维及多维数组

多维数组可以看作是一维数组的扩展。一个典型的二维数组可以看作是一个以一维数组为元素的特殊一维数组。当要存储和处理矩阵、图表等数据时,用二维数组更方便。二维数组的定义要指定两个下标范围,其语法形式为:

摇摇类型说明符摇变量名 [长度值 员] [长度值 圆];

习惯上,第一个下标称为行坐标,它的取值范围是 园~长度值 员原员;第二个下标是列坐标,其取值范围是 园~长度值 圆原员。例如,下面程序声明了一个二维整型数组,并通过一个二层循环语句给数组赋值:

```
摇摇裕include 约stdio.h 跃
摇摇摇摇void main ()
摇摇摇摇 {
摇摇摇摇摇摇int m [圆] [源];
摇摇摇摇摇摇int i, j;
摇摇摇摇摇摇for (i 越园; i 约圆; i 垣垣)
摇摇摇摇摇摇摇for (j 越园; j 约源; j 垣垣)
摇摇摇摇摇摇摇摇m [i, j] 越 i 垣 j;
摇摇摇摇}
```

二维数组元素在内存中是按行顺序存储的,第一个下标代表行,第二个下标代表列,这意味着按照在内存中的实际存储顺序访问数组元素时,右边的下标比左边的下标变化快一些。图 员鄄怨(葬) 是数组 皂所表示的矩阵,而图 员鄄怨(遭) 则是 皂的内存分布表示。

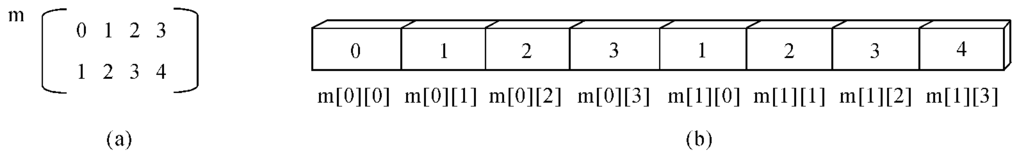

图 员鄄怨摇二维数组 皂的表示

二维数组的初始化与一维数组相似,为了达到图 员鄄怨的结果,可以利用下面的语句来初始化上面函数中的 皂:

摇摇摇int m {{园, 员, 圆, 猿}, {员, 圆, 猿, 源}};

多维数组的情况与二维数组类似,同样可以将其看作是一系列一维数组的嵌套。关于多维数组,需要注意一点:计算机要花大量时间计算数组下标,这意味着存取多维数组中的元素要比存取一维数组的元素花更多的时间。由于这些原因和其他原因,大量的多维数组一般采用 悦语言动态分配函数及指针的方法,每次对数组的一部分动态地分配存储空间。

员援远援猿摇指针

(员) 指针的基本概念

指针是 悦语言中最为灵活的一个因子,也是最难以掌握的一个概念。简单地说,指针就

是将内存地址作为值的变量，它与其他变量的不同之处在于，指针的存储空间存放的是地址，而不是一般的数据。

指针的物理意义十分明确，它是 悦语言的精华部分。有了指针技术，我们还可以描述复杂的数据结构，对字符串的处理可以更灵活，对数组的处理更方便，使程序的书写简洁、高效、清爽。通过利用指针，我们能很好地利用内存资源，使其发挥最大的效率。但难点在于它的使用极其灵活，几乎 悦语言的大多语法成分都能应用指针。

指针和前面已介绍的数组有着非常紧密的联系，一个数组一方面可以用其下标进行访问，另一方面也可以用指针来对它进行操作。

和其他变量类型一样，指针必须在使用前进行声明，在 悦语言中指针的声明只要在一般变量声明的变量名之前增加一个星号 ＊，其语法形式为：

摇摇类型说明符摇＊变量名；

例如，下面语句声明了两个指针：

```
摇摇char ＊cp 摇/＊字符指针变量 cp 可以存放字符变量单元的地址＊/
摇摇摇摇int ＊ip；   /＊整型指针变量 ip 可以存放整型变量单元的地址＊/
```

指针的两个基本运算是获取指针所指内存区域的数据和获取变量的地址，对应的运算符是＊和 驭。下列程序实现了简单的指针运算：

```
摇摇裕nclude 约stdio.h 跃
摇摇摇摇void main ()
摇摇摇摇{
摇摇摇摇摇摇char c员越'A';
摇摇摇摇摇摇char ＊cp；
摇摇摇摇摇摇cp 越&c员;
摇摇摇摇摇摇char c圆;
摇摇摇摇摇摇cp 越&c圆;
摇摇摇摇摇摇＊cp 越'B';
摇摇摇摇}
```

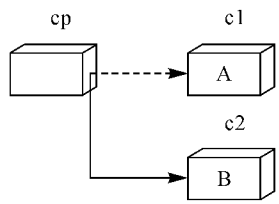

图 员源源摇指针的表示

执行此函数产生的结果可以用图 员源源形象地表示，初始时，糟员单元存放的是字符'粤'，指针变量 糟员存放的是 糟员单元的地址，用带箭头的线段表示 糟员所指的存储单元 糟员。接着定义另一字符变量 糟圆，并将其地址赋给 糟员，此时 糟员改变为指向 糟圆存储单元，原指向 糟员(虚线) 实际已不存在，最后一个语句通过运算符＊访问指针 糟员所指的存储单元 糟圆，并将其赋值为'月'。

有了指针，对于变量的访问不仅可以用变量名直接访问，同时也可以利用指针变量提供对变量的间接访问。例如在上面的例子中，最后一行对 糟圆的赋值就是一个间接访问。

对于指针变量可以进行加减操作，但是这里和普通数值变量的加减操作不同，指针变量的加减操作是以所指对象类型大小为单位的，下面的程序说明了这一点：

```
摇摇裕nclude 约stdio.h 跃
    摇摇void main ()
    摇摇{
    摇摇摇摇int a[源越{员，圆，猿，源};
```