21st CENTURY
规划教材

面向21世纪高等院校计算机系列规划教材

COMPUTER COURSES FOR UNDERGRADUATE EDUCATION

数据结构与算法

赵文静 主 编 祁 飞 副主编

科学出版社
www.sciencep.com

数据结构与算法

面向21世纪高等院校计算机系列规划教材
COMPUTER COURSES FOR UNDERGRADUATE EDUCATION

数据结构与算法

赵文静 主 编

祁 飞 副主编

科学出版社

北 京

内 容 简 介

本书是介绍“数据结构与算法”的教科书，首先介绍了有关算法的概念，然后系统地介绍了线性表、栈、队列、字符串、数组和广义表、树和二叉树、图等数据结构，并介绍了排序和查找的典型算法，同时还结合实例介绍了各种常用算法设计策略。

本书采用目前流行且广泛使用的C++语言作为算法描述与实现的工具，书中所有例程均在Microsoft Visual C++ 6.0调试通过。与本书配套的教学网站和教学用多媒体电子课件将会极大地方便读者。

本书适合作为计算机专业及相关专业的“数据结构”课程教材，对于参加各类计算机考试及自学计算机应用软件开发的读者也是一本合适的参考书。

图书在版编目(CIP)数据

数据结构与算法/赵文静主编. —北京：科学出版社，2005

(面向21世纪高等院校计算机系列规划教材)

ISBN 7-03-015050-3

Ⅰ.数… Ⅱ.赵… Ⅲ.①数据结构-高等学校-教材②数据分析-高等学校-教材 Ⅳ.TP.311.12

中国版本图书馆CIP数据核字(2005)第012814号

责任编辑：万国清 许 进 /责任校对：郝 岚

责任印制：吕春珉/封面设计：飞天创意

科学出版社 出版

北京东黄城根北街16号

邮政编码：100717

http://www.sciencep.com

双青印刷厂 印刷

科学出版社发行 各地新华书店经销

*

2005年8月第 一 版 开本：787×1092 1/16

2005年8月第一次印刷 印张：17 1/4

印数：1-3 000 字数：390 000

定价：23.00元

(如有印装质量问题，我社负责调换<路通>)

销售部电话 010-62136131 编辑部电话 010-62138978-8004(HI02)

前　言

数据结构与算法课程是计算机科学与技术专业的一门专业技术基础课，是计算机科学的算法理论基础和软件设计的技术基础，主要研究信息的逻辑结构及其基本操作在计算机中的表示和实现。对于从事计算机软件的设计开发、实现、测试与维护工作的人员，掌握基本的数据结构知识是非常必要的。

通过本书的学习，读者应该能够深刻理解各种常用的数据结构以及各种数据结构之间的逻辑关系，同时应能熟练掌握各种数据结构在计算机中的存储表示和在这些数据结构上的运算与实际的算法，并对算法的效率能够进行简要的分析。通过本书实例的学习，读者还能掌握复杂程序设计的方法，养成良好的编程习惯。

本书的特色在于：将算法实现的简单性与通用性相结合。在介绍各种数据结构时，先用 ADT（抽象数据类型）进行描述，然后用简单数据类型（比如整数）来实现该结构的基本运算算法，最后在应用实例中用 C++ 的模板类给出“泛型”算法的实现及使用例程。这样既兼顾了“可读性”，又实现了数据结构中数据类型的通用性，使读者学会用“泛型”简化设计，避免重复劳动，提高编程效率。

为了方便教学，按照教材的内容，我们制作好了对应的多媒体课件、所有书中程序源代码以及习题的参考答案。需要者请与作者联系，我们会免费提供给教材的使用者。另外，与该教材配套的教学网站也已开通，希望能够对使用者提供较大的便利。邮箱地址：qifei@xauat.edu.cn。

本书可作为计算机各专业及相关专业的“数据结构”课程教材，也可供从事计算机应用软件开发的读者参考。

赵文静编写本书第 1～3 章并制作 PowerPoint 课件，祁飞编写第 4～8 章并制作 PowerPoint 课件，张翔编写第 9～10 章并制作 PowerPoint 课件。书中典型实例的动画演示由王茹用 Flash 完成。全书由赵文静统稿。

本书的配套课件可在科学出版社的网站免费下载（http://www.sciencep.com）；也可在配套的教学网站免费下载（http://jwc.xarat.edu.cn/Content/jpkc/dsweb/default.htm）。

在本书的编写中作者参考了大量国内外相关教材，在此向有关作者表示感谢。

由于作者水平有限，书中错误在所难免，恳请广大读者和同行批评指正。

目　录

第1章 概 论

1.1 数据结构概述

20世纪40年代，电子数字计算机的问世使得解决复杂的计算问题成为可能。早期，电子计算机的应用范围只局限于科学和工程计算，其处理的对象是纯数值性的信息。通常，人们把这类问题称为数值计算。

随着计算机科学技术的迅猛发展，计算机的应用已从传统的数值计算领域发展到各种非数值计算领域。当前，计算机已广泛地应用于情报检索、企业管理、系统工程等各个领域；与此相应，计算机的处理对象也从简单的纯数值性数据发展到一般的符号和具有一定结构的数据。

于是，我们面临的问题是：对于每一种应用领域的处理对象，如何选择合适的数据表示(结构)，如何有效地组织计算机存储，在此基础上又如何有效地实现对象之间的“运算”关系。传统的解决数值计算的许多理论、方法和技术已不能满足解决非数值计算问题的需要，必须进行新的探索。数据结构就是研究和解决这些问题的重要基础理论。因此，数据结构是一门研究非数值计算的程序设计问题中计算机的操作对象以及它们之间的关系和操作等问题的学科。“数据结构”课程已成为计算机类各专业的一门重要专业基础课。

1.2 什么是数据结构

我们先介绍一些与数据结构相关的术语，然后给出数据结构的定义。

(1) 数据(Data)是信息的载体，它能够被计算机识别、存储和加工处理。它是计算机程序加工的“原料”，而电子计算机则是加工处理数据(信息)的工具。例如，求解代数方程的程序中所处理的数据是整数和实数，而编译程序或文本编辑程序中加工的数据是字符串，如今计算机可以处理图像、声音等。因此它们也属于数据的范畴。

(2) 数据元素(Data Element)是数据的基本单位。有些情况下，数据元素也称为节点、顶点、元素、记录。

(3) 数据项(Data Item)是数据的不可分割的具有独立含义的最小单位，数据元素是数据项的集合。

(4) 数据对象(Data Object)是具有相同性质的数据元素的集合。

(5) 数据结构(Data Structure)是研究数据元素之间的相互关系，即数据的组织形式。虽然至今还没有一个关于数据结构的标准定义，但它一般包括以下三方面的内容：

① 数据元素之间的逻辑关系，也称为数据的逻辑结构(Logical Structure)。

② 数据元素及其关系在计算机存储器内的表示，也称为数据的存储结构(Storage

Structure)。

③ 数据的运算,即对数据施加的操作。

数据的逻辑结构是指数据元素和数据元素之间的逻辑关系,它与数据的存储无关,是从具体问题抽象出来的数学模型。

数据的存储结构是指逻辑结构在计算机存储器里的实现(亦称为映象),它是依赖于计算机的,我们只在高级语言的层次上讨论存储结构。

数据的运算是定义在数据的逻辑结构上的,每种逻辑结构都有一个运算的集合。例如,最常用的运算有:检索、插入、删除、更新、排序等。这些运算实际上是在抽象的数据上所施加的一系列抽象的操作,所谓抽象的操作,是指我们只知道这些操作是"做什么",而无须考虑"如何做"。只有确定了存储结构之后,我们才考虑如何具体实现这些运算。换句话说,算法的设计取决于数据的逻辑结构,算法的实现取决于数据的物理存储结构。

在不易产生混淆之处,我们将数据的逻辑结构简称为数据结构,数据的逻辑结构可以用图来表示,用圆圈(节点)代表数据元素,用节点间的连线代表数据元素间的关系。

根据数据元素之间关系的不同,可将数据的逻辑结构分为集合、线性结构、树、图四类。

集合:若结构中的数据元素之间除了同属于一个集合的关系外别无其他关系,则称这种结构为集合,参看图 1.1(a)。

线性结构:若结构中的数据元素之间存在一个对一个的前驱后继关系,则称这种结构为线性结构参看图 1.1(b)。

在此种结构下,有且仅有一个元素无前驱元素,有且仅有一个元素无后继元素,其余任何一个元素均有且仅有一个前驱元素和一个后继元素。

树:若结构中的数据元素除一个特殊数据元素没有直接前驱外,其余每个数据元素都有且仅有一个直接前驱,则称这种结构为树,参看图 1.1(c)。

图:若结构中的每个数据元素都可以有任意多个直接前驱和直接后继,则称这种结构为图(网),参看图 1.1(d)。

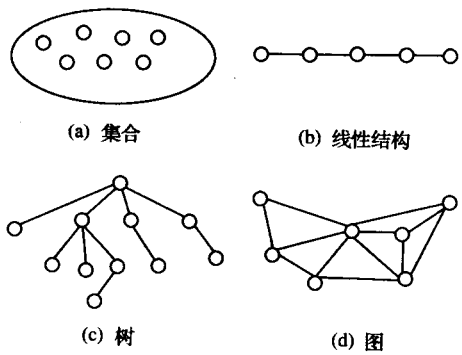

图 1.1 各种数据结构的图示

(6) 数据类型(Data Type)是指程序设计语言中各变量可取的数据种类,是在程序设计语言中已经实现了的数据结构。

(7) 抽象数据类型(Abstract Data Type)是指由用户定义,用以表示应用问题的数据模型。抽象数据类型由基本的数据类型构成,并包括一组相关的服务(操作);抽象数据类型简写作 ADT。

1.3 算 法

算法与数据结构密切相关,因为数据的运算是通过算法描述的,算法就是解决问题的策略、规则与方法。所以讨论算法是数据结构课程的重要内容之一。

1.3.1 什么是算法

简单地说,算法就是解决特定问题的方法。确切地说,就是对于某一类特定的问题,算法规定了一个运算过程(一系列操作),它必须满足下述准则:

① 输入:具有零个或多个输入的外界量,它们是算法开始前对算法最初给出的量。

② 输出:至少产生一个输出,它们是同输入有某种关系的量。

③ 有穷性:一个算法必须在执行有穷步之后结束,且每一步都可在有限时间内完成。

④ 确定性:算法的每一操作的含义都必须明确,无二义性。

⑤ 可行性:算法中每一操作,都是能够由计算机执行的。

1.3.2 算法的评价

求解同一类问题,可以有许多不同的算法,那么如何来评价这些算法的优劣呢?通常可从以下几个方面来衡量:

(1) 正确性(Correctness)。算法应该是"正确的",即对任何合法的输入,算法都能够得到正确的输出。

(2) 可读性(Readability)。算法应易于理解,即在正确性满足的前提下,越简单越好。

(3) 健壮性(Robustness)。算法应能识别非法的输入并能做出处理,而不是产生误动作或者陷入瘫痪。

(4) 时间复杂度(Time Complexity)。执行算法所耗费的时间的度量,即算法的时间性能。

(5) 空间复杂度(Space Complexity)。执行算法所耗费的存储空间的度量,主要考虑辅助存储空间。

当然,在性能(1)~(3)得以保证的情况下,我们希望选用一个所占存储空间小、运行时间短的算法。然而,实际上很难做到,原因是时间与空间的要求往往相互抵触,是一对矛盾。要节约算法的执行时间往往要以牺牲更多的空间为代价;而为了节省空间又可能要以更多的时间作代价。因此我们只能根据具体情况有所侧重。对于反复多次使用的程序,应尽可能选用快速的算法。若待解决的问题数据量极大,机器的存储空间较小,则相应算法主要考虑如何节省空间。

下面我们对算法的时间复杂度和空间复杂度的分析加以描述,首先介绍有关概念。

语句频度(Frequency Count)是指语句被重复执行的次数。即在算法中某一语句被重

复执行了 n 次，则其语句频度为 n。

一般情况下，算法中各语句的频度是问题规模 n 的某个函数 f(n)，算法的时间量度记作：

$$T(n) = O(f(n)) \tag{1.1}$$

它表示随问题规模 n 的增大，算法执行时间的增长率和 f(n)的增长率相同，称为算法的渐近时间复杂度(Asymptotic Time Complexity)，简称时间复杂度。请看例 1.1。

【例 1.1】 求两个 n 阶方阵的乘积 C=A×B，其算法的梗概如下：

```
# define max 100
  void MatrixMult(int n, float a[max][max], float b[max][max], float c[max]
                  [max])
  { int i, j, k; float x;
①  for(i = 1; i <= n; i ++)// n + 1
②  { for(j = 1; j <= n; j ++)// n(n + 1)
③   { x = 0;  // n²
④      for(k = 1; k <= n; k ++)// n²(n + 1)
⑤        x = x + a[i][k] * b[k][j];// n³
⑥   c[i][j] = x;// n²
    } //end_for j
    } //end_for i
  }
```

其中右边列出的是各语句的频度。语句①的循环控制变量 i 要增加到 n+1，测试 i>n 成立才会终止，故它的频度是 n+1，但是它的循环体却只能执行 n 次。语句②作为语句①循环体内的语句应该执行 n 次，但语句②本身要执行 n+1 次，所以语句②的频度是 n(n+1)。同理可得到语句③，④和⑤的频度分别是 n^2，$n^2(n+1)$，n^3。

该算法中所有语句的频度之和(即算法的时间耗费)为：

$$T(n) = 2n^3 + 4n^2 + 2n + 1 \tag{1.2}$$

由此可知，算法 MatrixMult 的时间耗费 T(n)是矩阵阶数 n 的函数。该算法的时间复杂度 T(n)当 n 趋向无穷大时，有：

$$\lim_{n\to\infty} T(n) = \lim_{n\to\infty} \frac{2n^3 + 3n^2 + 2n + 1}{n^3} = 2$$

当 n 充分大时，T(n)和 n^3 之比是一个不等于零的常数、即 T(n)和 n^3 是同阶的，或说 T(n)和 n^3 的数量级相同，记作 $T(n) = O(n^3)$。我们称 $T(n) = O(n^3)$是算法 MatrixMult 的渐近时间复杂度。其中记号“O”是数学符号，其严格的数学定义是：

若 T(n)和 f(n)是定义在正整数集合上的两个函数，当存在两个正的常数 c 和 n_0，使得对所有的 $n \geqslant n_0$，都有 $T(n) \leqslant cf(n)$成立，则 $T(n) = O(f(n))$。

当我们评价一个算法的时间性能时，主要标准是算法时间复杂度的数量级，即算法的渐近时间复杂度。例如，设有两个算法 A_1 和 A_2，求解同一问题，它们的时间复杂度分别是 $T_1(n) = 100n$，$T_2(n) = n^2$，当输入量 $n < 100$ 时，有 $T_1(n) > T_2(n)$，后者花费的时间较少。

但是，随着问题规模 n 的增大，两个算法的时间开销之比 $n^2/100n$ 亦随着增大。也就是说，当问题规模较大时，算法 A_1 比算法 A_2 要有效得多，它们的渐近时间复杂度 $O(n)$ 和 $O(n^2)$，正是从宏观上评价了这两个算法在时间方面的质量。因此，在算法分析时，往往对算法的时间复杂度和渐近时间复杂度不予区分，而经常是将渐近时间复杂度 $T(n)=O(f(n))$ 简称为时间复杂度，其中的 $f(n)$ 一般是算法中频度最大的语句频度。例如，我们说：算法 MatrixMult 的时间复杂度是 $T(n)=O(n^3)$，这里的 $f(n)=n^3$ 是该算法中语句⑤的频度。

按数量级递增排列，常见的时间复杂度有常数阶 $O(1)$，对数阶 $O(\log_2 n)$，线性阶 $O(n)$，线性对数阶 $O(n\log_2 n)$，平方阶 $O(n^2)$，立方阶 $O(n^3)$，…，k 次方阶 $O(n^k)$，指数阶 $O(2^n)$，即

$$O(1)\leqslant O(\log_2 n)\leqslant O(n\log_2 n)\leqslant O(n^2)\leqslant O(n^3)\leqslant \cdots \leqslant O(n^k)\leqslant O(2^n) \quad (1.3)$$

显然，时间复杂度为指数阶 $O(2^n)$ 的算法效率极低，在 n 值稍大时就无法应用。

类似于算法的时间复杂度，本书中以空间复杂度(Space Complexity)作为算法所需存储空间的量度，记作

$$S(n)=O(f(n)) \quad (1.4)$$

其中 n 为问题的规模(或大小)。一个上机执行的程序除了需要存储空间来寄存本身所用指令、常数、变量和输入数据外，也需要一些对数据进行操作的工作单元和存储一些为实现计算所需信息的辅助空间。如果输入数据所占空间只取决于问题本身，和算法无关，则只需要分析除输入和程序之外的额外空间；否则应同时考虑输入本身所需空间(和输入数据的表示形式有关)。若额外空间相对于输入数据量来说是常数，则称此算法为原地工作，第 8 章讨论的有些排序算法就属于这类。又如果所占空间量依赖于特定的输入，则除特别指明外，均按最坏情况来分析。

1.3.3 算法的描述

设计一个算法后，要对它进行描述，可以用自然语言、数字语言或约定的符号来描述，也可以用计算机高级程序语言来描述。本书选用 C++ 语言描述数据结构及算法；对每种数据结构先用抽象数据类型(ADT)简单给出这种数据结构的定义及基本运算，并用类 C 语言讲解一些典型的基本运算(操作)，在每章的末尾用 C++ 的类声明给出该数据结构的数据及操作(接口部分)，并且给出典型操作的实现代码(经过调试的算法)；ADT 是在概念层上描述问题；类是在实现层上描述问题；在应用层上操作对象(类的实例)解决问题。接口中声明了但没有给出算法的操作(运算、函数)由读者自行完成。C++ 语言的有关知识见相应的参考书目。

1.3.4 常用算法设计策略

常用的算法设计策略有：

(1) 递归技术(recursion method)——最常用的算法设计思想，与问题的可计算性密切相关，该思想体现于许多优秀算法之中。

(2) 分治法(Divide and Conquer method)——将一个难以直接解决的大问题，分割成一

些规模较小的相同问题,以便各个击破,分而治之;体现了一分为二的哲学思想。

(3) 动态规划(Dynamic Programming)——常用的求解决策过程(Decision Process)最优化问题的解决方法。

(4) 贪心算法(Greedy Algorithms)——采用贪心策略的算法设计。从问题的某一个初始解出发逐步逼近给定的目标,以尽可能快的地求得更好的解。当达到某算法中的某一步不能再继续前进时,算法停止。

(5) 回溯法(Backtracking Algorithm)——也称为状态空间搜索法,被称为"万能算法"的算法设计策略。

(6) 分枝-限界方法(branch and bound algorithm)——类似于回溯法,都是一种在问题的解空间树上搜索问题解的算法,但求解目标不同。

我们将在后面各章节结合应用实例具体介绍这些方法。

1.4 小 结

本章主要介绍了有关数据和数据结构的概念,数据结构在计算机科学中的重要性,而算法的描述则简要介绍了常用的算法设计策略以及从时间和空间角度分析算法的方法。

本章的学习要点:

1. 熟悉各名词和术语的含义;掌握各种基本概念,特别是数据结构的三个方面以及它们的相互关系。

2. 熟悉 C++ 语言的书写规范。

3. 了解算法的 5 个要素(准则)。

4. 掌握估算算法的时间复杂度的方法。

1.5 习 题

1. 什么是数据?它与信息是什么关系?

2. 什么是数据结构?有关数据结构的讨论涉及哪三个方面?

3. 什么是算法?算法的 5 个要素是什么?试根据这些特性解释算法与程序的区别。

4. 设 n 为正整数,利用大"O"记号将下列程序段的执行时间表示为 n 的函数。

①
```
i=1,k=100;
while(i<k)
  { k=k+1;   i+=10;}
```

②
```
for(i=1;i<=n;i++)
  for(j=1;j<=i;j++)
    for(k=1;k<=j;k++)x+=delta;
```

③
```
i=1;j=0;
while(i+j<=n)
if(i>j)j++;
 else i++;
```

④
```
x=91;y=100;
while(y>0)
if(x>100)
  { x-=10;y--;}
```

5. 设有数据结构(D,R),其中:

$D=\{d_1,d_2,\cdots,d_6\}$ $R=\{r\}$

$r=\{<d_1,d_3>,<d_1,d_5>,<d_2,d_4>,<d_2,d_5>,<d_3,d_6>,<d_5,d_6>\}$

请画出其逻辑结构图。

6. 已知 k 阶裴波那契序列的定义为

$f_0=0,f_1=0,\cdots,f_{k-2}=0,f_{k-1}=1;$

$f_n=f_{n-1}+f_{n-2}+\cdots+f_{n-k},n=k,k+1,\cdots$

试编写求 k 阶裴波那契序列的第 m 项值的函数算法，k 和 m 均以参数的形式在参量表中出现。

7. 按增长率由小至大的顺序排列下列各函数：

$2^{100},(3/2)^n,(2/3)^n,(4/3)^n,n^n,n^{3/2},n^{2/3},n!,n,\log_2 n,n/\log_2 n,n\log_2 n$

8. 试设定若干 n 值，比较两函数 n^2 和 $50n\log_2 n$ 的增长趋势，并确定 n 在什么范围内，函数 n^2 的值大于 $50n\log_2 n$ 的值。

9. 试用数学归纳法证明：

① $\sum_{i=1}^{n} i^2=\frac{n(n+1)(2n+1)}{6}$

② $\sum_{i=1}^{n} x^i=\frac{x^{n+1}-1}{x-1}(x\neq 1,n\geqslant 0)$