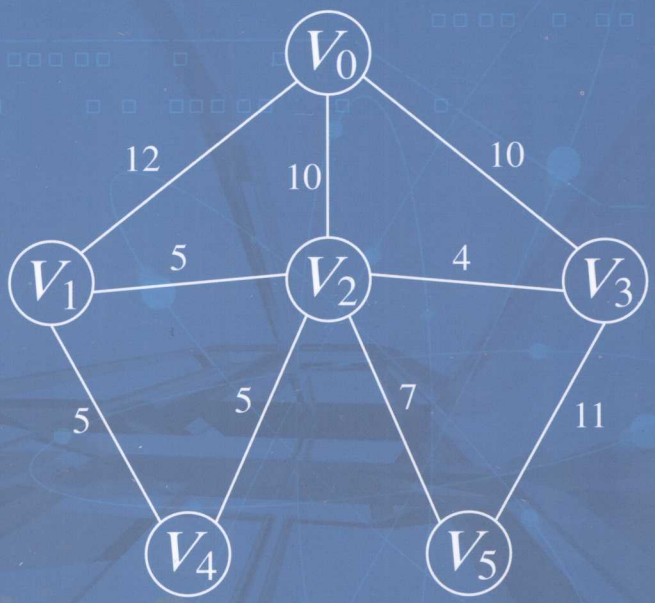
21 世纪高等学校规划教材

数据结构

黄同成　黄俊民　董建寅　编著

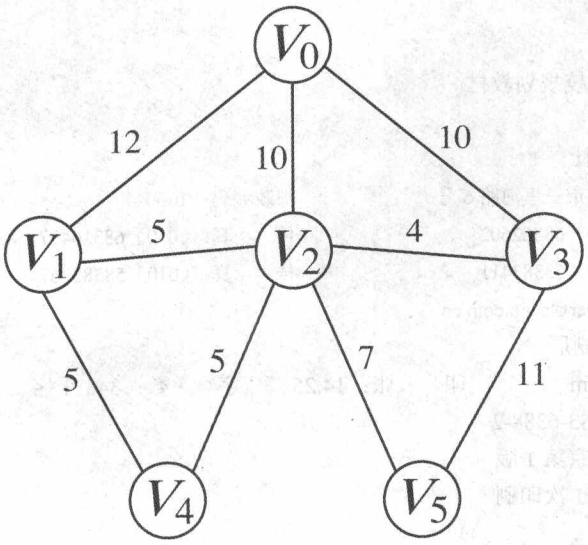


中国电力出版社
www.infopower.com.cn

21 世纪高等学校规划教材

数据结构

黄同成　黄俊民　董建寅　编著



中国电力出版社
www.infopower.com.cn

内容提要

“数据结构”是计算机学科的必修课程，本教材是作者针对数据结构课程概念多、算法灵活和抽象性强等特点，在总结长期教学经验的基础上编写而成的。全书共分 9 章，内容涵盖数据结构的基本概念、线性表、栈和队列、数组和稀疏矩阵、广义表和串、树和二叉树、图、排序、文件。每章后附有章节概括总结和习题。

本书内容丰富，层次分明，讲解深入浅出，可作为高等院校计算机及相关专业本科数据结构课程教材，也可供从事计算机软件开发与应用的工程技术人员参考。

图书在版编目（CIP）数据

数据结构 / 黄同成，黄俊民，董建寅编著. —北京：中国电力出版社，2008.2

21 世纪高等学校规划教材

ISBN 978-7-5083-6388-2

Ⅰ. 数… Ⅱ. ①黄… ②黄… ③董… Ⅲ. 数据结构－高等学校－教材 Ⅳ. TP311.12

中国版本图书馆 CIP 数据核字（2007）第 192908 号

丛 书 名：21 世纪高等学校规划教材

书　　名：数据结构

出版发行：中国电力出版社

地　　址：北京市三里河路 6 号　　邮政编码：100044

电　　话：（010）68362602　　传　　真：（010）68316497，88383619

服务电话：（010）58383411　　传　　真：（010）58383267

E-mail：infopower@cepp.com.cn

印　　刷：北京市同江印刷厂

开本尺寸：185mm×260mm　　印　　张：14.25　　字　　数：348 千字

书　　号：ISBN 978-7-5083-6388-2

版　　次：2008 年 2 月北京第 1 版

印　　次：2008 年 2 月第 1 次印刷

印　　数：0001—4000 册

定　　价：23.00 元

前　言

“数据结构”是计算机学科的必修课程，它涵盖了计算机学科的算法设计、操作系统、编译原理和数值分析等课程所涉及的大部分相关算法的实现。学习好该课程，不仅对这些后续课程的学习有很大帮助，而且在实际中有广泛的应用。

计算机是进行数据处理的工具。数据结构主要研究数据的各种组织形式及建立在这些结构之上的各种运算的实现。它不仅为使用程序设计语言进行编程提供了方法性的理论指导，还在一个更高的层次上总结程序设计的常用方法与技巧。

本教材是作者根据数据结构课程概念多、算法灵活和抽象性强等特点，在总结长期教学经验的基础上编写而成的。全书共分 9 章，第 1 章为“绪论”，介绍了数据结构的基本概念，特别强调算法分析的方法与技巧；第 2 章为“线性表”，介绍线性表的顺序与链式存储结构、逻辑结构及基本运算的实现过程；第 3 章为“栈和队列”，介绍栈与队列两种特殊的线性结构的概念与应用；第 4 章为“数组和串”，介绍了多维数组、稀疏矩阵、广义表和串的概念、相关运算及其实现过程；第 5 章为“树和二叉树”，介绍了树和二叉树的概念和各种运算的实现过程，其中特别突出了二叉树的各种递归算法实现；第 6 章为“图”，介绍了图的概念及各种运算算法的实现过程；第 7 章为“排序”，介绍了内排序和外排序的各种常用算法的实现与应用；第 8 章为“查找”，介绍了各种查找算法的实现过程；第 9 章为“文件”，介绍了各类文件的组织结构。

“数据结构”是一门应用性与实践性很强的课程，学生在掌握各种数据结构特别是存储结构的基础上，一定要尽可能多地上机实习，通过较多的实验把难以理解的抽象概念转化为实实在在的计算机能够正确运行的程序。只有这样才能将所学的知识和实际应用紧密结合起来，吸取算法的设计思想与精髓，提高运用这些知识解决实际问题的能力。因此，本教材突出上机实习内容，除最后一章外，其余各章都给出了相应的上机实验题，供教师和学生选用。

为了便于学习和上机实验，我们还编写了与本教材配套的《数据结构上机指导与题解》，构成了一个完整的教学系列。本系列中所有程序均在 Visual C++环境下调试通过。

本教材和配套指导书的编写得到了邵阳学院和上海金融学院领导及相关部门的大力支持，是两校课程组许多老师多年来在数据结构课程教学研究与教学改革中的经验与成果的结晶。本书由黄同成、黄俊民、董建寅编著。第 1、8 章由黄俊民编写，第 2 章由谢文平编写，第 3 章由成亚辉编写，第 4 章由董建寅编写，第 5 章由黄同成编写，第 6 章由丁竞渊编写，第 7、9 章由曾文飞编写，黄同成教授统稿。

由于水平所限，尽管编著者不遗余力，仍可能存在错误与不足，恳请广大读者批评指正。

编　者

2007 年 10 月

目　录

第1章 绪 论

半个多世纪以来，计算机技术的快速发展，使得计算机的应用从单纯的科学计算拓展到数据处理的领域。计算机需要表示、存储和处理的对象由纯粹的数值，发展到字符、表格、声音、图形和图像等各种各样的数据，而且数据量越来越大，组织结构越来越复杂。这给计算机程序设计带来许多困难。于是，在计算机科学与技术学科中出现了一门专业基础课程，专门研究对上述各种各样数据的抽像和归类后形成的少数几种最基本数据结构，阐述这些数据结构的内在逻辑关系，讨论它们在计算机中的存储形式，研究关于这些数据结构适用的各种基本操作（如查找、插入、删除、修改等）的算法。这门课程就是“数据结构”。数据结构是设计编译程序、操作系统、数据库系统，以及其他程序设计的重要基础。

1.1 什么是数据结构

数据是描述客观事物的、能输入到计算机中并被识别和处理的符号的集合。我们经常用到的数据有数值、字符、文字、声音、图形、图像、动画、影视等。

在计算机处理日常应用问题时，总是将数据按其性质归类到某个数据对象的集合中。在数据对象中，所有的数据元素都具有相同的性质。例如，整数数据对象的集合 $N=\{0, \pm1, \pm2, \cdots\}$；英文字母数据对象的集合 $A=\{'a', 'b', \cdots, 'z'\}$。除了系统定义的基本类型的数据对象外，还可有计算机用户自构建的复合的数据对象。例如，大型超市的商品信息，就是一种自构建的复合的数据对象。在这数据对象的集合中，每一个数据元素都记录了某一种商品的一组信息，它包括了商品号、商品名、单位、数量、单价等数据项。这些数据项可以分成两类，一类是初等项，如商品号、商品名、单位等，它们是数据处理时的最小单位，不能再被分割；另一类是组合项，如数量，它还可被分割为当前数量、最高库存量、临界数量等多个初等项。在实际应用中，这种自构建的复合的数据对象中的数据元素（也称为“记录”），被当作一个基本单位进行访问和处理。

一个数据对象的集合中，所有数据元素之间一定存在某种关系。例如，超市商品信息的数据对象中的记录，就是按商品号的由小到大顺序有序排列。又如，计算机文件系统的目录数据对象中的成员，形成一个层次结构，顶层是根目录，它的下一层是若干个子目录和若干文件目录，每个子目录的下一层还是若干个子目录和若干文件目录，依次类推，而文件目录则没有下一层，这些数据形成了一种树型关系。显然，研究某个数据对象的集合，必须要连带研究集合中所有数据元素之间的关系。

数据结构（Data Structure）是由某一数据对象及该对象中所有数据元素之间的关系组成。数据结构研究的是各种抽象的数据对象中的数据之间的逻辑关系（逻辑结构）和数据在计算机中的存储方式（物理结构），同时研究这些数据结构上的基本操作及其实现。

数据之间的逻辑关系，也称为逻辑结构，是指从解决问题的需要出发，为实现必要的功能

所抽象出来的数学模型。它面向问题，反映客观事物之间的逻辑关系，体现人们对某个特定领域的认识。当然，这些认识是在用户视图的层面上的，只是部分地反映为数字计算机领域中的数据间的逻辑关系和逻辑结构。

数据在计算机中的存储方式，也称为数据的物理结构，是指数据应该怎样在计算机中存放。它面向计算机，建立了由数据的逻辑结构到存储结构的一个映射。这个映射是在实现视图的层面上的，需要考虑具体问题的响应速度、处理时间、修改时间、存储空间和单位时间的处理量等要求。

专家认为，“程序设计”的技术，实际上就是“算法”加上“数据结构”。在各特定的数据逻辑结构和物理结构上，讨论相关的基本操作（算法）及其实现，分析算法的效率，也是数据结构的重要内容。

数据结构作为计算机科学技术学科的重要专业基础课，始于 1968 年。这一年，高德纳（Donald E. Knuth，“高德纳”是他在 1977 年起的中文名）在对计算机软件中的编译程序、属性文法和运算法则等进行深入研究的基础上，写作出版了经典的《计算机程序设计艺术》的第一卷，提出了算法及数据结构的概念。高德纳的《计算机程序设计艺术》出版，立刻成为计算机软件开发的经典之作，该书被翻译成几十种文字，在全世界广为流传。高德纳是计算机软件设计的重要奠基人。

1.2 基本概念和术语

为了本课程进一步讨论具备共同的基础，本节将对有关数据结构的基本概念和常用术语给出明确的含义。

首先，引进关于数据的几个概念。

所谓数据（Data），它是信息的载体，是客观事物的符号表示。在计算机科学中，是指所有能输入到计算机中并被计算机程序处理的符号的集合。例如，整数、实数、字符串，以及表示图像或声音的二进制数等，都可成为信息的载体，因而可被用来作为数据。

数据是计算机程序的工作对象。但是，由于各程序的功能不同，处理数据的内容和类型也不同。为了便于进一步讨论，我们将性质相同的数据归为一个集合，称为数据对象（Data Object）。数据对象是数据的子集。例如，我们有整数数据对象、字母字符数据对象等。数据对象的数据也可以是复合而成，由用户自定义。例如，上节介绍的超市商品数据对象。

组成数据对象集合的元素，称为数据元素（Data Element）。数据元素是数据的基本单位，在计算机程序中通常作为一个整体进行考虑和处理。例如，一个整数数值是整数数据对象中的一个元素，超市中某一种商品信息是超市商品数据对象的一个元素。数据对象，又称为记录（Record）、结点（Node）等。

一个数据元素可由若干个数据项（Data Item）组成。数据项可以分成两类，一类是初等项，如商品号、商品名、单位等，它们不能再被分割；另一类是组合项，如数量，它还可被分割为当前数量、最高库存量、临界数量等多个初等项。

初等项是数据中具有独立含义的、不可再分割的最小数据单位。它是客观实体的某一种特征的数据表示。在不会产生歧义的场合，数据项指的就是初等项。

其次，介绍数据结构的概念。

数据结构，是某一数据对象及该对象中所有数据元素之间的关系组成，记为{D，R}。其中，D是某一数据对象，R是该对象中所有数据元素之间的关系的有限集合。例如，在计算机处理中，复数可采用如下的数据结构：

Complex＝{C，R}

其中，数据对象C：{(c1，c2)｜c1和c2为实数}；R是定义在C上的一种关系，c1是复数的实部，c2是复数的虚部，各数据元素（c1，c2）可在复平面上得到体现。

数据结构研究的是各种抽象的数据对象中的数据之间的逻辑关系（逻辑结构）和数据在计算机中的存储方式（物理结构），同时研究这些数据结构上的基本操作及其实现（算法）。

再次，介绍数据的逻辑结构的一些概念。

在任何具体问题中，数据对象中的数据元素都不是孤立存在的，它们之间存在着某种关系。而这些关系与计算机系统无关，也与机器实现无关。数据的逻辑结构，就是指数据对象中的数据元素之间逻辑关系的抽象描述。

根据逻辑关系的性质不同，通常表现为四种基本的逻辑结构，如图1-1所示。

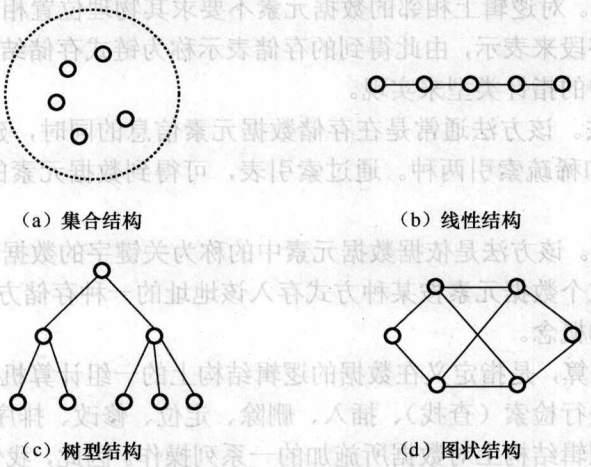


图1-1 四类基本结构的示意图

（1）集合结构。数据元素之间未定义任何关系的松散集合。例如，一个班级学生在操场中集合听广播，而学生之间的站立没有任何规定。

（2）线性结构。数据元素之间定义了次序关系的集合（全序集合），描述的是1对1关系。例如，同样是一个班级学生在操场中集中听广播，而学生之间的站立是按照学号次序站立或按照身高次序排队。

（3）树型结构。数据元素之间定义了层次关系的集合（偏序集合），描述的是1对多关系。它非常类似于自然界中的树，具有层次感，比如部队中人员关系由军、师、团、营、……的层次关系来划分。

（4）图状结构，或网状结构。数据元素之间定义了网状关系的集合，描述的是多对多关系。如交通网络图就是一个图状结构。

数据逻辑结构可分为线性结构和非线性结构两类。集合结构、树型结构和图状结构，统称为非线性结构。

然后介绍数据的物理结构（亦称存储结构）的一些概念。

计算机的主存储器的存储空间提供了一种具有非负整数地址编码的且相邻的单元集合，其基本的存储单元是字节。计算机的指令具有按地址随机访问存储空间内任意单元的能力。访问不同地址所需的访问时间基本相同。于是，需要研究如何将数据的逻辑关系映射到计算机主存储器的存储空间上，表现为数据的地址关系，而且这种表现方式要使得存储空间的开销小，且相应的算法效率高。

所谓数据的物理结构（存储结构），就是指数据逻辑结构在计算机主存储器中的具体实现。数据存储结构与孤立数据的表示形式不同，数据对象中的数据元素不但要表示其本身的实际内容，还要清楚地表示数据元素之间的逻辑关系。

数据存储主要有下列四种基本方法：

（1）顺序存储方法。是把逻辑上相邻的数据元素存储在物理位置相邻的存储单元中，由此得到的存储表示称为顺序存储结构。顺序存储结构是一种最基本的存储表示方法，通常借助于程序设计语言中的数组来实现。

（2）链式存储方法。对逻辑上相邻的数据元素不要求其物理位置相邻，数据元素间的逻辑关系通过附设的指针字段来表示，由此得到的存储表示称为链式存储结构。链式存储结构通常借助于程序设计语言中的指针类型来实现。

（3）索引存储方法。该方法通常是在存储数据元素信息的同时，建立附加的索引表。索引表一般有稠密索引和稀疏索引两种。通过索引表，可得到数据元素的存储地址，从而访问该元素。

（4）散列存储方法。该方法是依据数据元素中的称为关键字的数据项，直接计算出该元素的存储地址，而后将这个数据元素按某种方式存入该地址的一种存储方法。

最后，介绍算法的概念。

算法，即数据的运算，是指定义在数据的逻辑结构上的一组计算机操作的集合。例如，对于各种数据逻辑结构进行检索（查找）、插入、删除、定位、修改、排序等操作。要注意的是，这些运算实际上是在逻辑结构上对数据所施加的一系列操作。因此，我们虽然可以使用不同语言来描述算法，但是它们一种抽象的表述，并不依赖具体的计算机。

但是，运算的实现是依赖于所选取的存储结构，依赖于具体的计算机系统，依赖于不同的计算机程序设计语言的。

1.3 算法与算法分析

1.3.1 算法及其描述

算法（Algorithm）是为某一个特定问题而制定的求解步骤的一种描述，它是有限的指令序列，其中每一条指令表示一个或多个操作。一个算法应当具有下列重要特性：

（1）**输入**。一个算法必须有 0 个或多个输入。它们可以使用输入语句由外部提供，也可以使用置初值语句或赋值语句在算法内给定。这些输入取自于特定的对象的集合。

（2）**输出**。一个算法应有一个或多个输出，输出的量是算法计算的结果。

（3）**确定性**。算法的每一步都应确切地、无歧义地定义。在任何条件下，算法只有唯一的

一条执行路径，对于相同的输入只能得出相同的输出。

（4）**有穷性**。一个算法无论在什么情况下都应在执行有穷步后结束，且每一步都可在有穷时间内完成。

（5）**有效性**。算法中每一条指令都必须是能够被人或机器确切执行的基本指令，它们原则上都能够精确地执行，甚至人们仅用笔和纸做有限次运算就能完成。另外，指令的执行结果和结果的数据类型是能够预期的。

算法的描述可以有多种方式，如自然语言方式、程序语言方式、图形方式、表格方式等。以插入排序算法为例，我们可用自然语言来描述的算法如下：

假设待排序的 n 个结点的序列为 a_0，a_1，…，a_{n-1}，则依次对 $i=1$，2，…，$n-1$ 分别执行插入步骤：

假设前 i 个结点 a_0，a_1，…，a_{i-1} 已排序，有 $a_0 \leqslant a_1 \leqslant \cdots \leqslant a_{i-1}$，$t$ 是中间变量。将 a_i 送 t，然后将 t 依次与 a_{i-1}，a_{i-2}，…进行比较，将比 t 大的结点依次右移一个位置，直到某个 $j(0 \leqslant j \leqslant i-1)$ 有 $a_j \leqslant t$，则把 t 送原来 a_{j+1} 的位置；如果这样的 a_j 不存在，那么 a_0，a_1，…，a_{i-1} 都右移一个位置，此时将 t 送原来 a_0 的位置。

经过 $n-1$ 遍执行上述插入步骤，则 n 个结点的排序就完成了。

同样的插入排序，我们可用 C 程序语言来定义算法，具体见［算法 1-1］。

【算法 1-1】 插入排序算法。

```
void insertion(int a[ ], int n)
{
   int i, j;
   int t;
   for(i=1; i<n; i++) {                          // 执行 n-1 遍插入步骤
      t=a[i];
      for(j=i-1; j>=0 && t<a[j]; j--)            // 查找插入 t 的位置
         a[j+1]=a[j];
      a[j+1]=t;                                  // 将 t 插到适当位置
      }
}
```

本书的算法一般采用 C 程序语言描述，它的优点是程序紧凑、语句精练、程序结构化程度高、可读性好、通用性强。为了更好地交代算法的思路，在程序的注释中大量使用自然语言来描述算法。

判断一个算法的优劣，主要有以下几个标准：

（1）**正确性**。算法能够正确地执行预先规定的功能和性能的要求。这里包括对问题的求解要求的正确理解和程序设计语言对算法的正确实现，这是算法的最重要的标准。

（2）**可使用性**。也称为用户友好性，算法要能够很方便地使用。为此，算法要具有良好的界面，完备的用户文档；算法的设计必须符合抽象数据类型和模块化的要求，最好所有的输入和输出数据都通过参数表显式传递，少用公用变量或全局变量，每个算法只完成一个功能。

（3）**可读性**。算法应当是可读的，可理解的，这便于对其进行测试和修改。为此，算法的逻辑必须清晰、简单和结构化；所有的变量名、函数名的命名必须有实际含义且便于人们望文生义；在算法中必须加入注释，简要说明算法的功能、输入与输出参数的使用规则、重要数据的作用、算法中各程序段完成的功能等。

（4）**健壮性**。一个完整的算法对于不合理的数据，能适当地作出反应或进行纠错处理，而不会产生莫名其妙的输出结果。算法在输入参数、打开文件、读文件记录、子程序调用的操作时应有自动检错、报错并通过与用户对话来纠错的功能。纠错的方法应是返回一个表示错误或错误性质的值，而不是打印错误信息或异常并终止程序的执行，以便在更高的抽象层次上进行处理。

（5）**高效率和低存储量需求**。高效率的另一种说法就是算法执行时间短。对于同一个问题若有多个算法可求解，那么执行时间短的算法效率高。存储量需求就是指算法执行过程中所需要的最大存储空间。低存储量的算法将受到重视和推广。当然，执行时间和存储量都与问题的规模有关，问题规模越大，相对的执行时间就越长，需要的存储空间就越多。

1.3.2 算法分析和算法复杂度

一个算法与运行的环境有关，同样的算法在速度不同的计算机上运行，执行速度相差却非常大。一个算法用不同的编译器编译出来的目标代码也不一样长，完成同样的功能所需时间也不同。对于一个存储需求极大的算法，如果可用的存储空间不够，在运行时将不得不频繁地进行内外存交换，需要的运行时间就很多，如果可用的存储空间足够大，运行时间就明显地减少。由此可知，算法的运行时间依赖于所用的计算机系统、编译器、可用存储空间大小等。因此，一个算法的优劣，要在不同的机型、不同的编译器版本、不同的硬软件配置情况下通过测量后，根据实测结果作出判断，这是行不通的。

那么，评价算法的优劣，只有通过与具体运行环境和编译器版本无关的复杂性度量来进行比较。算法复杂性的度量，主要分为空间复杂度（Space Complexity）度量和时间复杂度（Time Complexity）度量。空间复杂度度量和时间复杂度度量，通过对算法的分析和事前估计的方法进行。

空间复杂度，是指当问题的规模以某种单位从 1 增加到 n 时，解决这个问题的算法在执行时所占用的存储空间也以某种单位由 1 增加到 $f(n)$，则称此算法的空间复杂度为 $f(n)$。空间单位一般规定为一个工作单元所占用的存储空间大小，这里的工作单元根据问题的要求可以是一个简单变量，也可以是一个构造型变量。

时间复杂度，是指当问题的规模以某种单位从 1 增加到 n 时，解决这个问题的算法在执行时所耗费的时间也以某种单位由 1 增加到 $T(n)$，则称此算法的时间复杂度为 $T(n)$。时间单位一般规定为一个程序步。程序步是指在语法上或语义上有意义的一段指令序列，而且这段指令序列的执行时间与实例特性无关。例如，简单变量的赋值语句，或者不包含函数调用的表达式运算，可认为程序步数为 1。

要确定一个程序的准确的程序步数是非常困难的，而且也不是很必要的。再说程序步数这个概念本身不是一个精确的概念，例如，赋值语句 $x=a$ 和 $x=a-b/(c+d)+e*f$，它们的程序步数都认为是 1，显然不很科学。因此，我们只需要分析算法的内部结构，找出关键的操作。考虑到程序步数与关键操作的执行频度是一一对应的，计算程序步数，可以转化为统计关键操作的执行频度，找出它与问题规模 n 的函数关系 $T(n)$，这就是时间复杂度。例如，在前面介绍的［算法 1-1］中，语句 $a[j+1]=a[j]$就是一个关键操作。我们不必统计所有的程序步数，只要统计这语句的执行频度，就可以作为这算法的时间复杂度。例如，［算法 1-1］中，作为关键操作的语句 $a[j+1]=a[j]$在最坏的情况下，需要执行 $\frac{n(n-1)}{2}$ 次，那么该算法在最坏情况下的

时间复杂度为$\frac{n(n-1)}{2}$。

假设求解某一个给定的问题有两种算法A和B，且它们时间复杂度分别为$T_A(n)$和$T_B(n)$，其中n表示所求问题的规模，当已知问题规模$n=n_0$时，有$T_A(n_0)\leqslant T_B(n_0)$，那么，我们认为算法A在规模为$n_0$时要优于算法B。假若对于一切的$n\geqslant 0$，都有$T_A(n)\leqslant T_B(n)$，那么，我们认为算法A优于算法B，且与问题的规模无关。

但在通常情况下，事先我们既不知道问题的规模，也不知道在整个问题规模的区间上一个函数是否小于或者等于另一个函数。在多数情况下，只要得到一个时间复杂度的估计值就足够了。为了研究算法的效率，我们考虑在问题规模的区间的大致方向上，时间复杂度函数的渐进特性（Asymptotic Behavior）。

假设算法的时间复杂度函数为$T(n)$，其中n为问题的规模，$f(n)$是一个函数，如果存在一个整数$n_0>0$和一个正整数C，对于任意$n\geqslant n_0$，都有$T(n)\leqslant c\,f(n)$，那么称$T(n)$是$f(n)$的大O表示，记为$T(n)=O(f(n))$。它表示随问题规模n的增大，算法时间复杂度的增长率和$f(n)$的增长率相同。函数$f(n)$是由我们选择的简单函数，如$f(n)=1$，$f(n)=\log_2 n$，$f(n)=n$，$f(n)=n^2$，……，那么，算法的时间复杂度的比较，实际上是表示为简单函数的渐进时间复杂度的比较。

例如，下列三个程序段：

（1）`{++x; s = 0;}`

（2）`for (i=0; i<n; ++i) {++x; s += x;}`

（3）
```
for (i=0; i<n; ++i)
    for(j=0; j<n; ++j) {++x; s += x;}
```

其中，“++x”为各段的关键操作，它们在各段中的执行频度分别为1、n和n^2，则三个程序段的时间复杂度分别为$O(1)$、$O(n)$和$O(n^2)$。

根据大O表示定义，有下列性质：

$$O(k\cdot f_1(n))=O(f_1(n))$$

$$O(f_1(n)+f_2(n))=O(\max(f_1(n),\ f_2(n)))$$

$$O(f_1(n)\cdot f_2(n))=O(f_1(n))\cdot O(f_2(n))$$

其中，所谓$\max(f_1(n),\ f_2(n))$是指当n充分大时取$f_1(n)$、$f_2(n)$中的大值，在这个意义上显然有下列关系：

$$c<\log_2 n<n<n\cdot\log_2 n<n^2<n^3<\cdots<n^k<\cdots<2^n<3^n<n!$$

其中，c是与n无关的任意正数，k为正整数。如果一个算法的时间复杂度为前面几个函数的大O表示，如c、$\log_2 n$、n，$n\cdot\log_2 n$，那么它的时间效率比较高；如果取到n^2、n^3，其时间效率差强人意；如果取到2^n、3^n、$n!$，那么当n稍大时，算法的时间代价就会变得很大，没有实用价值。

对于［算法1-1］的最坏情况下的时间复杂度$\frac{n(n-1)}{2}$，其渐进时间复杂度$T(n)$为

$$O\left(\frac{n(n-1)}{2}\right)=O(n^2-n)=O(n^2)$$

本章概括与总结

本章介绍了关于数据结构的基本概念。数据结构包含逻辑结构、存储结构和运算三个方面的内容。要了解线性结构与非线性结构之间的区别，了解线性结构、树型结构、图状结构之间的差别，了解顺序存储、链式存储、索引存储和散列存储等方法之间的差别，重点掌握算法的时间复杂度和空间复杂度的分析。

习 题

1. 简述数据、数据对象与数据元素的关系与区别。
2. 数据结构主要研究三个方面的问题，分别是什么？
3. 数据的逻辑结构分为两大类，它们是什么？
4. 算法的五个重要特性是什么？
5. 分别举出生活中线性结构、树型结构、图状结构的数据模型实例。
6. 分析下面程序段的时间复杂度。

(1)

```
for(i=1;i<=n;i++)                                    //①
{
    s++;                                             //②
    for(j=1;j<=2*n;j++)                              //③
    {t++;}                                           //④
}
```

(2)

```
for(i=1;i<=n;i++)                                    //①
    for(j=1;j<=n;j++)                                //②
        {
          c[i][j]=0;                                 //③
          for(k=1;k<=n;k++)                          //④
            c[i][j]+=a[i][k]*b[k][j];                //⑤
        }
```

(3)

```
int a[]={10,2,9,7,3,6,4,1}
 order(int j,int m)
 {
   int i,temp;
   if(j<m)
   {
     for(i=j+1;i<=m;i++)
       if(a[i]<a[j])
         {
           temp=a[i];
           a[i]=a[j];
           a[j]=temp;
         }
     j++;
     order(j,m);
   }
 }
```

第2章 线 性 表

线性表是最简单、最基本，也是最常用的一种线性结构。线性结构的特点是，在数据元素的非空有限集中：

（1）存在唯一的一个被称为“第一个”的数据元素。

（2）存在唯一的一个被称为“最后一个”的数据元素。

（3）除第一个之外，集合中的每个数据元素均只有一个前驱。

（4）除最后一个之外，集合中每个数据元素均只有一个后继。

线性表有两种存储方法：顺序存储和链式存储，它的主要基本操作是插入、删除和检索等。

2.1 线性表的定义和基本运算

2.1.1 线性表的定义

线性表是一种线性结构。线性结构的特点是数据元素之间是一种线性关系，数据元素“一个接一个地排列”。在一个线性表中数据元素的类型是相同的，或者说线性表是由同一类型的数据元素构成的线性结构。在实际问题中线性表的例子是很多的，如学生情况信息表是一个线性表：表中数据元素的类型为学生类型；一个字符串也是一个线性表：表中数据元素的类型为字符型等等。

综上所述，线性表定义如下：

线性表是具有相同数据类型的 $n(n\geqslant 0)$ 个数据元素的有限序列，通常记为：

$$(a_1, a_2, \cdots, a_{i-1}, a_i, a_{i+1}, \cdots, a_n)$$

其中 n 为表长，$n=0$ 时称为空表。

表中相邻元素之间存在着顺序关系。将 a_{i-1} 称为 a_i 的直接前趋，a_{i+1} 称为 a_i 的直接后继。也就是说，对于 a_i，当 $i=2, \cdots, n$ 时，有且仅有一个直接前趋 a_{i-1}，当 $i=1, 2, \cdots, n-1$ 时，有且仅有一个直接后继 a_{i+1}，而 a_1 是表中第一个元素，它没有前趋，a_n 是最后一个元素，无后继。

需要说明的是：a_i 是序号为 i 的数据元素（$i=1, 2, \cdots, n$），通常我们将它的数据类型抽象为 datatype，datatype 根据具体问题而定。

在稍复杂的线性表中，一个数据元素可以由若干个数据项组成。在这种情况下，常把数据元素称为记录，含有大量记录的线性表又称为文件。如在学生情况信息表中，它是用户自定义的学生类型；在字符串中，它是字符型等等。

2.1.2 线性表的基本运算

在第 1 章中提到，数据结构的运算是定义在逻辑结构层次上的，而运算的具体实现是建立

在存储结构上的。因此，下面定义的线性表的基本运算作为逻辑结构的一部分，每一个操作的具体实现只有在确定了线性表的存储结构之后才能完成。

线性表上的基本操作有：

（1）线性表初始化：Init_List(L)。
初始条件：线型表 L 不存在。
操作结果：构造一个空的线性表。

（2）求线性表的长度：Length_List(L)。
初始条件：线性表 L 存在。
操作结果：返回线性表中的所含元素的个数。

（3）取表元：Get_List(L,i)。
初始条件：线性表 L 存在且 1≤i≤Length_List(L)。
操作结果：返回线性表 L 中的第 i 个元素的值或地址。

（4）按值查找：Locate_List(L,x)，x 是给定的一个数据元素。
初始条件：线性表 L 存在。
操作结果：在表 L 中查找值为 x 的数据元素，其结果返回在 L 中首次出现的值为 x 的那个元素的序号或地址，称为查找成功；否则，在 L 中未找到值为 x 的数据元素，返回一特殊值(0)表示查找失败。

（5）插入操作：Insert_List(L,i,x)。
初始条件：线性表 L 存在，插入位置正确（1≤i≤n+1，n 为插入前的表长）。
操作结果：在线性表 L 的第 i 个位置上插入一个值为 x 的新元素，使原序号为 i，i+1，…，n 的数据元素的序号变为 i+1，i+2，…，n+，插入后表长=原表长+1。

（6）删除操作：Delete_List(L,i)。
初始条件：线性表 L 存在，1≤i≤n。
操作结果：在线性表 L 中删除序号为 i 的数据元素，删除后使序号为 i+1，i+2，…，n 的元素变为序号为 i，i+1，…，n-1，新表长=原表长-1。

需要说明的是：

（1）某数据结构上的基本运算，不是它的全部运算，而是一些常用的基本的运算，还可进行一些更复杂的操作。例如，将两个或两个以上的线性表合并成一个线性表；把一个线性表拆开成两个或两个以上的线性表等。而每一个基本运算在实现时也可能根据不同的存储结构派生出一系列相关的运算来。比如线性表的查找在链式存储结构中还会有按序号查找；再如插入运算，也可能是将新元素 x 插入到适当位置上等等，不可能也没有必要全部定义出它的运算集，读者掌握了某一数据结构上的基本运算后，其他的运算可以通过基本运算来实现，也可以直接去实现。

（2）在上面各操作中定义的线性表 L 仅仅是一个抽象在逻辑结构层次的线性表，尚未涉及到它的存储结构，因此每个操作在逻辑结构层次上尚不能用具体的某种程序语言写出具体的算法，而算法的实现只有在存储结构确立之后。

【例 2-1】假设利用两个线性表 *La* 和 *Lb* 分别表示两个集合 A 和 B（即线性表中的数据元素即为集合中的成员），现要求一个新的集合 $A=A\cup B$。这就要求对线性表作如下操作：扩大线性表 *La*，将存在于线性表 *Lb* 中而不存在于 *La* 中的数据元素插入到线性表 *La* 中去。只要从线性表 *Lb* 中依次取得每个数据元素，并依值在线性表 *La* 中进行查访，若不存在，则插入之。上述操作过程可用以下算法描述。

```
void union(List &La,List Lb){
/*将所有在线性表 Lb 中但不在 La 中的数据元素插入到 La 中*/
La_len= Length_List(La); Lb_len= Length_List(Lb); /*求线性表的长度*/
```

```
for(i=1;i<=Lb_len;i++) {
e=Get_List(Lb,i); /*取 Lb 中第 i 个数据元素赋给 e*/
if(!Locate_List(La,e)) Insert_List(La,++La_len,e);}
}
```

[例 2-1] 的时间复杂度为 O(Length_List(La)×Length_List(Lb))。

【例 2-2】 已知线性表 La 和 Lb 中的数据元素按值非递减有序排列，现要求将 La 和 Lb 归并为一个新的线性表 Lc，且 Lc 中的数据元素仍按值非递减有序排列。例如，设

La=（3，5，8，11）

Lb=（2，6，8，9，11，15，30）

则 Lc=（2，3，5，6，8，8，9，11，11，15，30）

从上述问题要求可知，Lc 中的数据元素或是 La 中的数据元素，或是 Lb 中的数据元素，则只要先设 Lc 为空表，然后将 La 或 Lb 中的元素逐个插入到 Lc 中即可。为使 Lc 中元素按值非递减有序排列，可设两个指针 i 和 j 分别指向 La 和 Lb 中某个元素，若设 i 当前所指的元素为 a，j 当前所指的元素为 b，则当前应插入到 Lc 中的元素 c 为

$$c=\begin{cases} a & \text{当} a \leqslant b \text{时} \\ b & \text{当} a > b \text{时} \end{cases}$$

显然，指针 i 和 j 的初值均为 1，在所指元素插入 Lc 之后，在 La 或 Lb 中顺序后移。上述归并算法如下所示：

```
void MergeList(List La,List Lb,List &Lc){
Init_List(Lc); i=j=1;k=0;
La_len= Length_List(La); Lb_len= Length_List(Lb);
while((i<=La_len)&&(j<=Lb_len){ /*La 和 Lb 均非空*/
ai= Get_List(La,i); bj= Get_List(Lb,j);
if(ai<=bj) { Insert_List(Lc,++k,ai);++i;}
else { Insert_List(Lc,++k,bj);++j;}}
while(i<=La_len) { ai= Get_List(La,i);i++; Insert_List(Lc,++k,ai);}
while(j<=Lb_len) { bj= Get_List(La,i);j++; Insert_List(Lc,++k;bj);}}
```

上述算法的时间复杂度为 O(Length_List(La)+Length_List(Lb))。

2.2 线性表的顺序存储结构

2.2.1 线性表顺序存储的概念

线性表的顺序存储是指在内存中用地址连续的一块存储空间顺序存放线性表的各元素，用这种存储形式存储的线性表称其为顺序表。因为内存中的地址空间是线性的，因此，用物理上的相邻实现数据元素之间的逻辑相邻关系是既简单又自然的，如图 2-1 所示。设 a_1 的存储地址为 $Loc(a_1)$，每个数据元素占 d 个存储地址，则第 i 个数据元素的地址为

$$Loc(a_i)=Loc(a_1)+(i-1)\times d \quad 1\leqslant i\leqslant n$$

这就是说只要知道顺序表首地址和每个数据元素所占地址单元的个数就可求出第 i 个数据