高等学校教材

数据结构
——用C语言描述

唐策善　李龙澍　黄刘生　编著

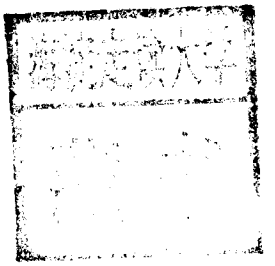
高等教育出版社

高等学校教材

数据结构

——用C语言描述

唐策善　李龙澍　黄刘生　编著



高等教育出版社

内 容 提 要

本书系统地介绍了各种常用的数据结构以及排序、查找的各种算法。阐述了各种数据结构的逻辑关系、存储表示及运算操作，并对C语言描述的算法作了详细的注解和简要的性能分析。全书既注重原理又注重实践，配有大量图表、例题和习题，内容丰富，概念讲解清楚，逻辑性强，可读性好。各章的小结可以使读者抓住本章重点。书中针对不同层次教学的特点和需要用“*”号标明。每章备有习题。

本书可作为高等院校计算机有关专业本科生、专科生的教材，亦可作为成人教育（面授或函授）的教材，还可供广大从事计算机应用的科技人员参考。

(京)112号

图书在版编目(CIP)数据

数据结构：用C语言描述/唐策善等编著.—北京：高等教育出版社，1995
高等学校教材
ISBN 7-04-005265-2

JS582/19

Ⅰ.数… Ⅱ.唐… Ⅲ.程序设计-数据结构 Ⅳ.TP311.12

中国版本图书馆CIP数据核字(95)第00389号

*

高等教育出版社出版
新华书店总店北京发行所发行
化学工业出版社印刷厂印装

*

开本 787×1092 1/16 印张 16.25 字数 430 000
1995 年 4 月第 1 版 1997 年 7 月第 3 次印刷
印数 18 101—27 110
定价 12.50 元

前　言

在计算机及其应用的各个领域中，都会用到各种各样的数据结构，学会分析研究计算机加工对象的特性，选择合适的数据结构和存储表示，以及编制相应的实现算法，是计算机工作者不可缺少的知识。因此“数据结构”是高等院校计算机专业教学中的一门技术基础课程，在我国当前的计算机专业教学计划中，它是核心课程之一。

本书介绍各种最常用的数据结构，阐述各种数据结构内涵的逻辑关系，讨论它们在计算机中的存储表示，以及在这些数据结构上的运算(操作)和实际的执行算法，并对算法的效率进行简要的分析和讨论。

全书包含十章内容和一个附录：前面四章主要介绍几种基本的数据结构，即线性表、栈、队列和串，它们均属于线性结构；第五至七章叙述非线性结构，它们是多维数组、广义表、树和图；第八、九两章分别介绍数据处理中广泛使用的技术——排序和查找；第十章讨论外存储器上的数据结构——文件；最后的附录概要地叙述C语言，便于读者理解书中用C语言描述的算法。

全书既注重原理又重视实践，配有大量的例题和习题，解释比较详细。书中算法采用C语言描述，且均在计算机上运行调通，又做了较详细的注解，有利于读者理解算法的实质和基本思想。书中每一章均有小结，概述了该章的内容提要和学习要求，以便于读者学习和抓住重点。每章配置的习题可以检验读者的学习效果和培养程序设计的能力。

本书既可作为计算机专业学生的教材，亦可供从事计算机应用的工程技术人员参考，尤其适合于那些使用C语言编程的科技人员。使用本书作为本科生教材时，其内容可以讲授一个学期；使用它作为专科生教材时，可以删去带＊的章节；使用它作为成人教育(面授或函授)的教材时，建议酌情再精减有关内容。

本书是在中国科学技术大学唐策善、黄刘生老师的《数据结构》讲义基础上编写而成。经过反复磋商和共同讨论，由李龙澍提供绝大多数修改稿，黄刘生提供少量的修改稿，经唐策善教授审阅和整理后定稿，因而本书是我们三位作者共同合作的产物。

本书初稿经过教学实践的检验，反映较好。初稿完成后，承蒙北京大学计算机系许卓群教授详细审阅，提出了许多宝贵意见；其他老师亦提供了许多有益的建议，作者谨此一并致以诚挚的谢意。

由于作者水平有限，书中缺点或错误在所难免，殷切希望广大读者批评指正。

唐策善　李龙澍　黄刘生

1994年12月于合肥

目　录

第一章 概 论

1.1 数据结构的概念

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"。例如,一个代数方程求解程序中所用的数据是整数和实数,而一个编译程序或文本编辑程序中使用的数据是字符串。随着计算机软、硬件的发展,计算机应用领域的扩大,数据的含义也随之拓广了。例如,当今计算机可以处理图象、声音等,因此它们也属于数据的范畴。

数据元素(Data Element)是数据的基本单位。有些情况下,数据元素也称为元素、结点、顶点、记录。有时一个数据元素可以由若干个数据项(也可称为字段、域)组成,数据项是具有独立含义的最小标识单位。

数据类型(Data Type)是具有相同性质的计算机数据的集合及在这个数据集合上的一组操作。如整数,它是[-maxint,maxint]区间上的整数(maxint 是依赖于所使用的计算机及语言的最大整数),在这个整数集上可以进行加、减、乘、整除、取模等操作。

数据类型可以分为原子数据类型和结构数据类型。原子数据类型是由计算机语言提供的,如 C 语言的整数型、字符型;结构数据类型是借用计算机语言提供的一种描述数据元素之间逻辑关系的机制,由用户自己定义的,如 C 语言的数组结构等。

数据结构(Data Structure)指的是数据之间的相互关系,即数据的组织形式。它一般包括以下三个方面的内容:

(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,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。

数据的逻辑结构是从逻辑关系上描述数据,它与数据的存储无关,是独立于计算机的。因此,数据的逻辑结构可以看作是从具体问题抽象出来的数学模型。数据的存储结构是逻辑结构用计算机语言的实现(亦称为映象),它是依赖于计算机语言的,对机器语言而言,存储结构是具体的,但我们只在高级语言的层次上来讨论存储结构。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。本书中讨论的数据运算,均以 C 语言描述的算法来实现。

为了增加对数据结构的感性认识,下面举例来具体说明上述概念。

例 1.1 学生成绩表,见表 1.1。

表 1.1 学生成绩表

学 号	姓 名	数学分析	普通物理	高等代数	平均成绩
880001	丁一	90	85	95	90
880002	马二	80	85	90	85
880003	张三	95	91	99	95
880004	李四	70	84	86	80
880005	王五	91	84	92	89
⋮	⋮	⋮	⋮	⋮	⋮

我们把表 1.1 称为一个数据结构，表中的每一行是一个结点(或记录)，它由学号、姓名、各科成绩及平均成绩等数据项组成。该表中数据元素之间的逻辑关系是：对表中任一个结点，与它相邻且在它前面的结点(亦称为**直接前趋**(Immediate Predecessor))最多只有一个；与表中任一结点相邻且在其后的结点(亦称为**直接后继**(Immediate Successor)) 也最多只有一个。表中只有第一个结点没有直接前趋，故称为开始结点；也只有最后一个结点没有直接后继，故称之为终端结点。例如，表中"马二"所在结点的直接前趋结点和直接后继结点分别是"丁一"和"张三"所在的结点，上述结点间的关系构成了这张学生成绩表的逻辑结构。

该表的存储结构则是指用计算机语言如何表示结点之间的这种关系，即表中的结点是顺序邻接地存储在一片连续的单元之中，还是用指针将这些结点链接在一起？在这张表中，可能要经常查看某一学生的成绩，当学生退学时要删除相应的结点，进来新学生时要增加结点。究竟怎样进行查找、删除、插入，这就是数据的运算问题。搞清楚了上述三个问题，也就弄清了学生成绩表这个数据结构。

综上所述，我们可以将数据结构定义为：按某种逻辑关系组织起来的一批数据，应用计算机语言，按一定的存储表示方式把它们存储在计算机的存储器中，并在这些数据上定义了一个运算的集合，就叫做一个数据结构。

在不会产生混淆的前提下，我们常常将数据的逻辑结构简称为数据结构。数据的逻辑结构有两大类：

(1) 线性结构

线性结构的逻辑特征是：有且仅有一个开始结点和一个终端结点，并且所有结点都最多只有一个直接前趋和一个直接后继。线性表就是一个典型的线性结构。本书第二章到第四章介绍的都是线性结构。

(2) 非线性结构

非线性结构的逻辑特征是一个结点可能有多个直接前趋和直接后继。第五章到第七章讨论的数据结构都是非线性结构。

数据的存储结构可用以下四种基本的存储方法得到：

(1) 顺序存储方法

该方法是把逻辑上相邻的结点存储在物理位置上相邻的存储单元里，结点间的逻辑关系由存储单元的邻接关系来体现。由此得到的存储表示称为**顺序存储结构**(Sequential Storage Structure)，通常顺序存储结构是借助于程序语言的数组来描述的。

该方法主要应用于线性的数据结构，非线性的数据结构也可以通过某种线性化的方法来

实现顺序存储。

(2) 链接存储方法

该方法不要求逻辑上相邻的结点在物理位置上亦相邻，结点间的逻辑关系是由附加的指针字段表示的。由此得到存储表示称为**链式存储结构**(Linked Storage Structure)，通常要借助于程序语言的指针类型来描述它。

(3) 索引存储方法

该方法通常是在存储结点信息的同时，还建立附加的索引表。索引表中的每一项称为索引项，索引项的一般形式是：(关键字，地址)，关键字是能唯一标识一个结点的那些数据项。若每个结点在索引表中都有一个索引项，则该索引表称为**稠密索引**(Dense Index)。若一组结点在索引表中只对应一个索引项，则该索引表称之为**稀疏索引**(Sparse Index)。稠密索引中索引项地址指出结点所在的存储位置，而稀疏索引中索引项的地址则指示一组结点的起始存储位置。

(4) 散列存储方法

该方法的基本思想是根据结点的关键字直接计算出该结点的存储地址。

上述四种基本的存储方法，既可以单独使用，也可以组合起来对数据结构进行存储映象。同一种逻辑结构采用不同的存储方法，可以得到不同的存储结构。选择何种存储结构来表示相应的逻辑结构，视具体要求而定，主要考虑是运算方便及算法的时空要求。

值得指出的是，很多教科书上是将数据的逻辑结构和数据的存储结构定义为数据结构，而将数据的运算定义为数据结构上的操作。但是，无论是怎样定义数据结构，都应该将数据的逻辑结构、数据的存储结构及数据的运算这三方面看成一个整体。希望读者学习时，不要孤立地去理解一个方面，而要注意它们之间的联系。

正是因为存储结构是数据结构不可缺少的一个方面，所以我们常常将同一逻辑结构的不同存储结构，冠以不同的数据结构名称来标识它们。例如，线性表是一种逻辑结构，若采用顺序方法的存储表示，则称该结构为顺序表；若采用链接方法的存储表示，则称该结构为链表；若采用散列方法的存储表示，则可称其为散列表。

同理，由于数据的运算也是数据结构不可分割的一个方面，在给定了数据的逻辑结构和存储结构之后，按定义的运算集合及其运算的性质不同，也可能导致完全不同的数据结构。例如，若对线性表上的插入、删除运算限制在表的一端进行，则该线性表称之为栈；若对插入限制在表的一端进行，而删除限制在表的另一端进行，则该线性表称为队列。更进一步，若线性表采用顺序表或链表作为存储结构，则对插入和删除运算做了上述限制之后，可分别得到顺序栈或链栈、顺序队列或链队列。

1.2 为什么要学习数据结构

数据结构是计算机软件和计算机应用专业的核心课程之一，在众多的计算机系统软件和应用软件中都要用到各种数据结构。因此，仅掌握几种计算机语言是难以应付众多复杂的课题，要想有效地使用计算机，还必须学习数据结构的有关知识。

在计算机发展初期，人们使用计算机主要是处理数值计算问题。由于当时所涉及的运算对象是简单的整型、实型或布尔型数据，所以程序设计者的主要精力是集中于程序设计的技巧

上，而无须重视数据结构。随着计算机应用领域的扩大和软、硬件的发展，“非数值性问题”越来越显得重要。据统计，当今处理非数值性问题占用了90%以上的机器时间，这类问题涉及到的数据结构更为复杂，数据元素之间的相互关系一般无法用数学方程式加以描述。因此，解决此类问题的关键已不再是分析数学和计算方法，而是能设计出合适的数据结构，才能有效地解决问题。

著名的瑞士计算机科学家沃思(N. Wirth)教授曾提出：算法＋数据结构＝程序。这里的数据结构是指数据的逻辑结构和存储结构，而算法则是对数据运算的描述。由此可见，程序设计的实质是对实际问题选择一种好的数据结构，加之设计一个好的算法，而好的算法在很大程度上取决于描述实际问题的数据结构。请看下面的两个例子。

例 1.2 电话号码查询问题。

假定要编写一个程序，查询某个城市或单位的私人电话号码。对任意给出一个姓名，若该人有电话号码，则要迅速地找到其电话号码；否则指出该人没有电话号码。解此问题首先要构造一张电话号码登记表，表中每个结点存放两个数据项：姓名和电话号码。要写出好的查找算法，取决于这张表的结构及存储方式。最简单的方式是将表中结点顺序地存储在计算机中，查找时从头开始依次查对姓名，直到找出正确的姓名或是找遍整个表均没有找到为止。这种查找算法对于一个不大的单位或许是可行的，但对一个有成千上万私人电话的城市就不实用了。然而，若这张表是按姓氏排列的，则我们可以另造一张姓氏索引表，采用如图 1.1 所示的存储结构。那么查找过程是先在索引表中查对姓氏，然后根据索引表中的地址到电话号码登记表中核查姓名，这样查找登记表时就不需查找其它姓氏的名字了。因此，在这种新的结构上产生的查找算法就更为有效。在查找一章中我们还要讨论有关的查找策略。

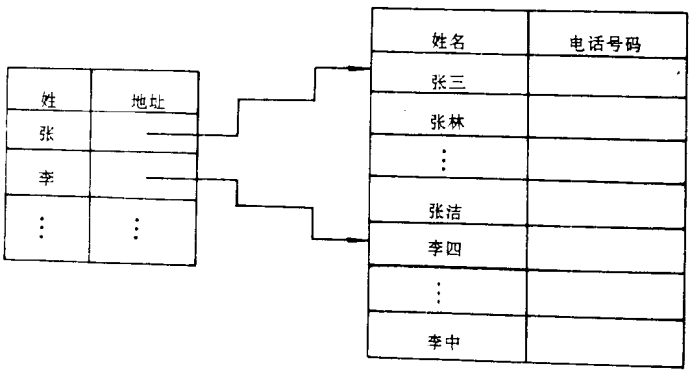


图 1.1 电话号码查询问题的索引存储

例 1.3 田径赛的时间安排问题。

假设某校的田径选拔赛共设六个项目的比赛，即跳高、跳远、标枪、铅球、100 米和 200 米短跑，规定每个选手至多参加三个项目的比赛。现有五名选手报名参赛，选手所选择的项目如表 1.2 所示。

现在要求设计一个竞赛日程安排表，使得在尽可能短的时间内安排完比赛。为了能较好地解决这个问题，首先应该选择一个合适的数据结构来表示它。为此，我们可以设计这样的一个图(见图 1.2)，图中顶点代表竞赛项目，在所有的两个不能同时进行比赛的项目之间连上一条

边。显然同一个选手选择的几个项目是不能在同一时间内比赛的，因此该选手所选择的项目中应该两两有边相连。由此可得到如图 1.2 所示的数据结构模型，图中 A,B,…,F 分别对应于六个竞赛项目。例如，丁一选择的三个项目是 A,B 和 E，因而 A 与 B 之间，B 与 E 之间均有边相连。

表 1.2　参赛选手比赛项目表

姓　名	项目 1	项目 2	项目 3
丁一	跳高	跳远	100 米
马二	标枪	铅球	
张三	标枪	100 米	200 米
李四	铅球	200 米	跳高
王五	跳远	200 米	

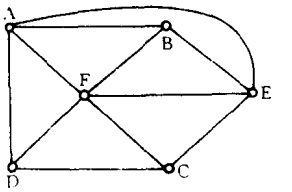


图 1.2　安排竞赛项目的数据结构模型

上述由顶点和边组成的无向图是数据结构中非线性数据结构中的一类。竞赛项目的时间安排问题可以抽象为对该无向图进行“着色”操作，即使用尽可能少的颜色去给图中每个顶点着色，使得任意两个有边连接的相邻顶点着上不同的颜色。每一种颜色表示一个比赛时间，着上同一颜色的顶点是可以安排在同一时间内竞赛的项目。例如，图中顶点 A 和 C 不相邻，可以选取颜色 1 为它们着色；同理，B 和 D 可着同一种颜色 2；E 和 F 有边相连，它们是相邻的，应该分别着上颜色 3 和颜色 4。也就是说，只要安排 4 个不同的时间竞赛即可。时间 1 内可以比赛跳高(A)和标枪(C)，时间 2 内可以比赛跳远(B)和铅球(D)，时间 3 和时间 4 内分别比赛 100 米(E) 和 200 米(F)。

从上述例子不难看出，解决问题的一个关键步骤是，选取合适的数据结构表示该问题，然后才能写出有效的算法。

1.3　算法描述

因为数据的运算是通过**算法**(Algorithm)描述的，所以讨论算法是数据结构课程的重要内容之一。

通俗地讲，一个算法就是一种解题方法。更严格地说，算法是由若干条指令组成的有穷序列，它必须满足下述准则：

(1) 输入：具有 0 个或多个输入的外界量，它们是算法开始前对算法给出的最初量。

(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：每一条指令的执行次数必须是有限的。

(4) 确定性：每条指令的含义都必须明确，无二义性。

(5) 可行性：每条指令的执行时间都是有限的。

在一个算法中，有些指令可能是重复执行的，因而指令的执行次数可能远远大于算法中的指令条数。由有穷性可知，对于任何输入，一个算法在执行了有限条指令后一定要终止；又由可

行性知道，一个算法必须在有限时间内完成。因此，一个程序如果对任何输入都不会陷入无限循环，则它就是一个算法。

算法的含义与程序十分相似，但二者是有区别的。一个程序不一定满足有穷性。例如，系统程序中的操作系统，只要整个系统不遭破坏，它就永远不会停止，即使没有作业要处理，它仍处于一个等待循环中，以待新作业的进入。因此操作系统就不是一个算法。另外，程序中的指令必须是机器可执行的，而算法中的指令则无此限制。但是一个算法若用机器可执行的语言来书写，则它就是一个程序。

一个算法可以用自然语言、数学语言或约定的符号语言来描述。本书用C语言来描述算法。

C语言的语法结构简单说明如下：

1. 函数的形式

[数据类型]函数名([形式参数])
[形式参数说明;]
{内部数据说明;
执行语句;
}

函数的定义主要由函数名和函数体两大部分组成。函数体用花括号"{"和"}"括起来，函数中用方括号括起的部分可以省略，但函数名后面的圆括号是不能省略的。将函数加工的结果传递到函数外的参数，必须用指针来实现。执行语句由一个或多个语句构成，两个语句之间用分号";"分隔开。函数通常结束于return(表达式)语句，结束时将表达式的值返回给调用者。若函数出现异常情况，其返回值由用户规定。

2. 条件语句有两种格式

(1) if (表达式) 语句;
(2) if (表达式) 语句1;
else 语句2;

3. 循环语句有三种格式

(1) while (表达式)
{ 循环体语句;
}

while循环首先计算表达式的值，若表达式的值非零，则执行循环体中语句，然后再次计算表达式的值。重复上述过程，直至表达式的值为零时退出循环，转去执行循环体下面的语句。

(2) do
{
循环体语句;
}while (表达式);

do-while循环首先执行循环体的语句，然后计算表达式的值。若表达式的值非零，则再次执行循环体，再计算表达式的值…，直至表达式的值为零时结束循环。

(3) for (表达式1;表达式2;表达式3)
{

```
    循环体语句;
}
```

首先计算表达式 1，然后求表达式 2 的值，若结果非零(非假)，则执行循环体语句，最后进行表达式 3 的运算。如此循环，直至表达式 2 的值为零。

在上述的循环体语句中，可使用 break 语句，其功能是控制程序从循环体中退出，从而结束循环过程。当循环体语句只有一条语句时，花括号“{”和“}”可以省略。

4. 情况语句

```
switch (表达式)
{
  case  判断值 1:
          语句组 1;
          break;
  case  判断值 2:
          语句组 2;
          break;
          …
  case  判断值 n;
          语句组 n;
          break ;
  [default :
          语句组;
          break;]
}
```

switch-case 首先计算表达式的值，然后用其结果与判断值比较，当它们一致时就执行该 case 下的语句组。若结果值与所有的判断值都不相同，则执行 default 下的语句组。和函数的定义类似，这里的方括号部分亦可以省略。

5. 赋值语句

(1) 变量=表达式;

它表示把表达式的值赋给变量。

(2) 变量++

它表示变量加 1 后赋值给变量。

(3) 变量--;

它表示变量减 1 后赋值给变量。

6. 输入、输出语句

(1) 字符:用函数 getchar 和 putchar 实现;

(2) 其它数据:用函数 scanf 和 printf 实现其输入和输出。

7. 注解形式

/* 字符串 */

1.4 算法分析

求解同一个问题，可以有许多不同的算法，究竟如何来评价这些算法的好坏呢？

显然，选用的算法首先应该是“正确的”。此外，主要考虑如下三点：

(1) 执行算法所耗费的时间；

(2) 执行算法所耗费的存储空间，其中主要考虑辅助存储空间；

(3) 算法应易于理解，易于编码，易于调试等等。

当然，我们希望选用一个所占存储空间小、运行时间短、其它性能也好的算法。然而，实际上很难做到十全十美。原因是上述要求有时相互抵触，要节约算法的执行时间往往要以牺牲更多的空间为代价；而为了节省空间可能耗费更多的计算时间。因此我们只能根据具体情况有所侧重。若该程序使用次数较少，则力求算法简明易懂；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。本书主要讨论算法的时间特性，偶尔也讨论空间特性。

一个算法所耗费的时间，应该是该算法中每条语句的执行时间之和，而每条语句的执行时间是该语句的执行次数(也称为**频度**(Frequency Count))与该语句执行一次所需时间的乘积。但是，当算法转换为程序之后，每条语句执行一次所需的时间取决于机器的指令性能、速度以及编译所产生的代码质量，这是很难确定的。我们假设执行每条语句所需的时间均是单位时间。一个算法的时间耗费就是该算法中所有语句的频度之和。于是，我们就可以独立于机器的软、硬件系统来分析算法的时间耗费。

例 1.4 求两个 n 阶方阵的乘积 C = A×B，其算法如下：

```
        #define n 自然数
        MATRIXMLT(A,B,C)
        float A[ ][n],B[ ][n],C[ ][n];
        { int i,j,k;
(1)     for (i=0;i<n;i++)                          n+1
(2)       for (j=0;j<n;j++)                        n(n+1)
(3)       { C[i][j]=0;                             n²
(4)         for(k=0;k<n;k++)                       n²(n+1)
(5)         C[i][j]=C[i][j]+A[i][k]*B[k][j];       n³
          }
        }/*MATRIXMLT*/
```

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，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)。同理可得语句(3)，(4)和(5)的频度分别是 n^2，$n^2(n+1)$和 n^3。该算法中所有语句的频度之和(即算法的时间耗费)为：

$$T(n)=2n^3+3n^2+2n+1 \tag{1.1}$$

由此可知，算法 MATRIXMLT 的时间耗费 T(n)是矩阵阶数 n 的函数。

一般地，我们将算法求解问题的输入量(或初始数据量)称为问题的**规模**(Size，大小)，并用一个整数表示。例如，矩阵乘积问题的规模是矩阵的阶数，而一个图论问题的规模则是图中的顶点数或边数。一个算法的**时间复杂度**(Time Complexity，也称时间复杂性)T(n)则是该算法的时间耗费，它是该算法所求解问题规模 n 的函数。当问题的规模 n 趋向无穷大时，我们把时间复杂度 T(n)的数量级(阶)称为算法的渐近时间复杂度。

例如，算法 MATRIXMLT 的时间复杂度 T(n)如(1.1)式所示，当 n 趋向无穷大时，显然有

$$\lim_{n\to\infty} T(n)/n^3 = \lim_{n\to\infty} (2n^3+3n^2+2n+1)/n^3 = 2 \tag{1.2}$$

这表明，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，或者说 T(n)和 n^3 的数量级相同，可记作 $T(n)=O(n^3)$。我们称 $T(n)=O(n^3)$ 是算法 MATRIXMLT 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 时，使得对所有的 $n\geqslant n_0$，都有 $T(n)\leqslant c\cdot f(n)$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 $T_1(n)=100n^2$，$T_2(n)=5n^3$，当输入量 $n<20$ 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 $5n^3/100n^2=n/20$ 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 $O(n^2)$和 $O(n^3)$，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，算法 MATRIXMLT 的时间复杂度一般是指 $T(n)=O(n^3)$，这里的 $f(n)=n^3$ 是该算法中语句(5)的频度。

下面再举几例，说明如何求算法的时间复杂度。

例 1.5 交换 a 和 b 的内容。

```
temp=i;
i=j;
j=temp;
```

以上三条单个语句的频度均为 1，该程序段的执行时间是一个与问题规模 n 无关的常数，因此，算法的时间复杂度为常数阶，记作 $T(n)=O(1)$。事实上，只要算法的执行时间不随着问题规模 n 的增加而增长，即使算法中有上千条语句，其执行时间也不过是一个较大的常数，此时，算法的时间复杂度也只是 O(1)。

例 1.6 变量计数之一。

```
(1) x=0; y=0;
(2) for (k=1; k<=n; k++)
(3)   x++;
(4) for (i=1; i<=n; i++)
(5)   for (j=1; j<=n; j++)
(6)     y++;
```

一般情况下，对步进循环语句只需考虑循环体中语句的执行次数，而忽略该语句中步长加1、终值判别、控制转移等成分。因此，以上程序段中频度最大的语句是(6)，其频度为 $f(n)=n^2$，

所以该程序段的时间复杂度为 $T(n)=O(n^2)$。由此可见，当有若干个循环语句时，算法的时间复杂度是由嵌套层数最多的循环语句中最内层语句的频度 f(n)决定的。

例 1.7 变量计数之二。

```
(1) x=1;
(2) for(i=1;i<=n;i++)
(3)   for (j=1;j<=i;j++)
(4)     for(k=1;k<=j;k++)
(5)       x++;
```

显然，该程序段中频度最大的语句是(5)，内循环的执行次数虽然与问题规模 n 没有直接关系，但是却与外层循环的变量取值有关，而最外层循环的次数直接与 n 有关，因此可以从内层循环向外层分析语句(5)的执行次数：

$$\sum_{i=1}^{n}\sum_{j=1}^{i}\sum_{k=1}^{j}1=\sum_{i=1}^{n}\sum_{j=1}^{i}j=\sum_{i=1}^{n}i(i+1)/2=[n(n+1)(2n+1)/6+n(n+1)/2]/2$$

则该程序段的时间复杂度为 $T(n)=O(n^3/6+\text{低次项})=O(n^3)$。

很多算法的时间复杂度不仅仅是问题规模 n 的函数，还与它所处理的数据集的状态有关。在这种情况下，通常是根据数据集中可能出现的最坏情况，估计出算法的**最坏**(Worst)**时间复杂度**。有时，我们也对数据集的分布作出某种假定(如等概率)，并讨论算法的**平均**(Average)**时间复杂度**。

例 1.8 在数组 A[n]中查找值为 K 的元素，若找到，则返回其位置 i(0≤i<n)；否则返回 -1，算法如下：

```
(1)  i=n-1;
(2)  while ( (i>=0) && (A[i] != K ) )
(3)  i--;
(4)  return i;
```

显然，此算法中语句(3)的频度不仅与问题规模 n 有关，还与 K 和数组 A 中各元素的取值有关。但是在最坏情况下，while 循环要执行 n 次，即语句(3)频度的最大值是 f(n)=n，因此该算法的最坏时间复杂度为 T(n)=O(n)。

将常见的时间复杂度，按数量级递增排列，则依次为：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 $O(2^n)$。图 1.3 展示了各种时间复杂度的增长率。显然，时间复杂度为指数阶 $O(2^n)$的算法效率极低，当 n 值稍大时就无法应用。

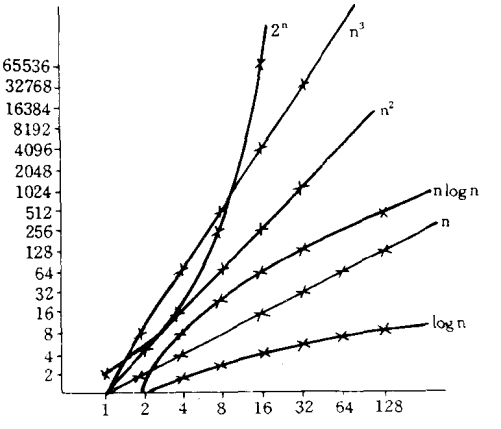


图 1.3 几种常见的函数增长率

类似于时间复杂度的讨论，一个算法的**空间复杂度**(Space Complexity)S(n)定义为该算法所耗费的存储空间，它也是问题规模 n 的函数。渐近空间复杂度也常常简称为空间复杂度。

小　结

本章简要地介绍了数据、数据结构等基本概念；阐述了数据结构所包含的三个方面的内容，即数据的逻辑结构、数据的存储结构和数据的运算；讨论了线性结构和非线性结构的逻辑特征，以及数据存储的四种基本方法。读者学习这些内容和例子后，希望能对数据结构的基本概念，具有初步的认识和理解。

本书使用C语言来描述算法，为了便于读者了解C语言描写的算法，本章简要介绍了C语言的程序结构和主要语句的语法。

算法和数据结构密切相关，不可分割，本章引进了算法、算法的时间复杂度等概念，以及分析时间复杂度的简易方法。

习　题

1.1 简述下列概念：

数据，数据元素，数据类型，数据结构，逻辑结构，存储结构，线性结构，非线性结构。

1.2 试举一个数据结构的例子，叙述其逻辑结构、存储结构、运算这三个方面的内容。

1.3 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=0;
while (i<n)
{k=k+10*i; i++;
}
```

(2)
```
i=0;k=0;
do
{k=k+10*i; i++;
}while (i<n);
```

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

(4)
```
x=n; /* n>1 */
while (x>=(y+1)*(y+1))
  y++;
```

(5)
```
x=91;y=100;
while (y>0)
  if (x>100){x=x-10;y--;}
  else x++;
```

1.4 比较C语言与Pascal语言的差异.

1.5 按增大率由小至大的顺序排列下列各函数：

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