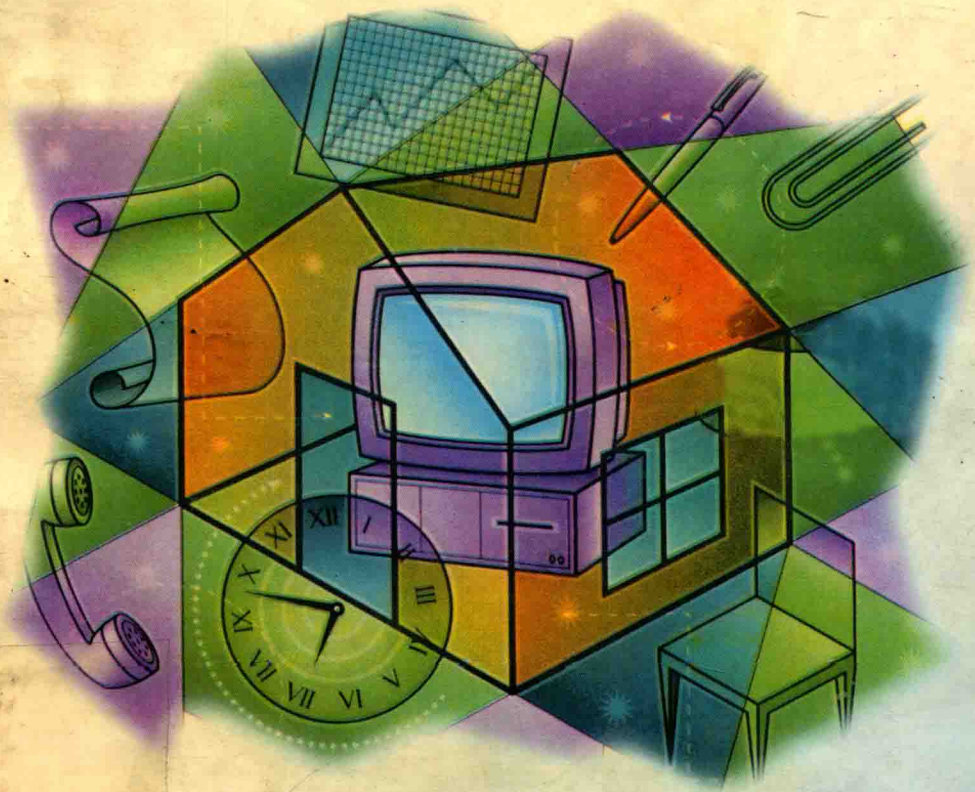
数据结构

陈本林　陈珮珮　吉根林　编著



南京大学出版社

数 据 结 构

陈本林　陈珮珮　吉根林　编著

南京大学出版社

内 容 提 要

本书第一章介绍数据类型和数据结构的基本概念；第二、三、四章讲述表、栈、队列和串等基本的抽象数据类型及其实现方法；第五、六章分别介绍树和图以及它们的应用；第七、八两章分别讲述各种内排序和查找算法；第九章简要介绍外排序算法。全书概念叙述清楚，语言流畅，有丰富的例题和习题。

本书可作为大学计算机专业学生的教材，也可供其他从事计算机工作的人员参考。

数 据 结 构

陈本林　陈珮珮　吉根林　编著

*

南京大学出版社出版

（南京大学校内　邮政编码：210093）

江苏省新华书店发行　　南京通达印刷厂印刷

*

开本：787×1092　1/16　印张：15.75　字数：390千

1998年6月第1版　　1998年6月第1次印刷

印数：1—4000

ISBN　7-305-03154-2/TP·177

定价：20.80元

前　言

本书是大学计算机专业和其他有关专业学生学习数据结构的教材。数据结构是计算机专业的一门基础课程，它是学习操作系统、编译原理、数据库系统等课程的重要基础。

全书共分九章，第一章介绍数据类型和数据结构的基本概念，如何利用抽象数据类型方法组织和设计程序；还介绍了算法的概念和算法分析的方法。第二章到第四章分别讨论表、栈、队列和串等线性结构，以及这些结构在动态存储管理、递归模拟和算术表达式编译等方面的应用。第五章和第六章讨论树和图两种非线性结构。第五章的内容主要包括树、二叉树、线索树等，举例说明树形结构在计算机科学和工程技术中的应用。第六章介绍表示图的数据结构以及图的一些重要算法。第七章介绍几种常用的内排序算法以及这些算法的分析。第八章讨论在线性结构和非线性结构上的查找算法，其中有顺序查找、二分法查找、分块查找、散列方法、二叉查找树、平衡二叉树、trie 结构、B 树和 B^+ 树。第九章简要介绍外排序方法。各章内都有丰富的例题，每一章之后配有相应的习题供课后练习和上机实习。

数据结构这门课程的主要任务应该是使学生学会怎样对处理的数据建立抽象数据类型，利用抽象数据类型进行程序设计；使学生学会用程序设计语言中提供的数据类型描述和定义数据结构来实现抽象数据类型。作者正是力求利用这一观点展开全书的内容。

在使用本书作为教材时，可视教学对象的具体情况和学时的多少适当加以取舍，一般可略去目录中标有“*”的章节。

本书各章内容由三人分工执笔，陈本林对全书进行了全面的整理、修改并定稿。

作者特别感谢南京大学计算机科学与技术系张福炎教授，他对本书的出版始终给予关心和支持。

作　者

1997 年 9 月

目　录

第一章　概　　论

用计算机解决一个特定问题，必须建立适合该问题的数学模型，依据这个模型找出问题的解法。起初，可以用自然语言或其它非形式的方法来描述求解的算法，然后逐步将问题形式化。在用某种程序设计语言编码时，需要根据问题的模型建立一些新的数据类型，用程序设计语言的过程形式表示这些新类型上的基本操作，以过程调用代替对新类型的操作。本章将讨论数据类型和数据结构的概念，以及算法和算法分析的内容。

1.1　数据类型和数据结构

一个**数据类型**(Data type)由一个**值集**和定义在这个值集上的一个**操作集**组成。例如，数据类型 integer(整型)的值集是{…，－2，－1，0，1，2，…}，其操作集包括加、减、乘、除以及求绝对值等等。大多数程序设计语言都提供一组固有的数据类型，这些数据类型通常分为两类：**原子数据类型**和**结构数据类型**。原子数据类型的值是单个的不可分的数据元素，PASCAL 语言中的布尔类型、字符类型、枚举类型以及若干数值类型都是原子类型。结构数据类型是用已有的数据类型构造的新的数据类型，PASCAL 语言中的记录类型、数组类型是结构类型。

结构类型的值由一组元素组成，下列数组类型 Sample：

```
type
    Sample=array[1..3] of real;
```

它的每个值由排成一行的三个实数组成，对任意一个值(0.5，2.1，3.4)，0.5 为第一个元素，2.1 为第二个元素，3.4 为第三个元素，元素之间有着固定的关系。值(0.5，2.1，3.4)和值(2.1，3.4，0.5)的组成元素相同，但元素间的顺序不同，因此是两个不同的值。

计算机存储器是由一系列二进位(bit)组成的，8 个二进位为一组，称为一个**字节**(byte)，每个字节有一个地址。程序中的“变量”是一种抽象形式。用高级语言编制程序时，程序员不必知道程序中各种类型的数据在存储器中的具体表示。例如，在图 1.1(a)中，程序员说明一个 integer 型变量 x 并给 x 赋值 735；编译程序在将高级语言的源程序转换成目标语言或机器语言程序时，确定与变量 x 相对应的内存位置或地址，并生成赋值的机器指令。目标程序执行时，完成对 x 的赋值。假定编译程序分配给 x 的两个字节的地址为 1246 和 1247，执行赋值的结果如图 1.1(b)所示。

```
var
    x:integer;{保留空间}
      .
      .
      .
    x:=735;
```

x: 0 0 0 0 0 0 1 0 1 1 0 1 1 1 1 1

(a) 程序员看到的整数赋值的抽象　　(b) 16bit 的一个内存位置(两个字节)存有一个二进制数

图 1.1　整数赋值的抽象与实现

在某些计算机的硬件指令集中,没有实现实数或浮点操作的指令,计算机系统将提供一组子程序来完成这些操作。当你写一个赋值语句 x:=x*y+3.0 时,语言编译程序不但要指出变量 x 和 y 的硬件存储位置,还必须产生对完成加法和乘法的子程序的调用。这是一类软件级的抽象。

从上述两个例子可以看出,通过使用**抽象**(abstraction)(利用变量说明和运算符),程序员不必为实现指定操作所需的内部表示和实际指令的细节而烦恼。抽象是达到信息隐蔽的一种方法,它将数据的表示和操作过程对用户隐蔽起来。

抽象数据类型(Abstract data type)提供组织和设计程序的一种新方法。抽象数据类型的基本思想是将数据类型的使用与它的**表示**和**实现**分开,通过把一个数据类型的表示及在这个数据类型上的操作局限在一个程序模块中将数据类型封装起来,用户只需了解怎样使用一个数据类型而不必知道它的实际表示和具体实现,从而减少了程序设计的复杂性。

一个抽象数据类型分成**说明**(specification)和**实现**(implementation)两个主要部分。说明部分定义一个应用程序怎样使用该抽象数据类型,它包括**语法**(syntax)和**语义**(semantics):语法指明所有操作的名称、操作所需的对象的数量、类型及返回值的类型;语义指明数据类型的每种操作的定义、对每个可能的输入将返回什么值。考虑具有操作 init、ismember、insert 和 delete 的数据类型 capital,我们用过程首部给出语法描述,用前置条件(pre)和后置条件(post)给出语义描述*:

```
procedure init (var S:capital);
    post  建立一个空集 S。
function ismember (x:char; S:capital): boolean;
    post  如果 x 为集合 S 中的元素,则 ismember 为 true,否则为 false。
procedure insert (x:char; var S:capital);
    pre   集合 S 未满且 x∉S。
    post  x 成为集合 S 的一个元素。
procedure delete (x:char; var S:capital);
    pre   x∈S。
    post  从集合 S 中删去元素 x。
```

抽象数据类型的实现包括**表示**(representation)和**算法**(algorithms)。表示指明抽象数据类型的值在内存中如何表示,在一个程序中利用语言提供的数据类型建立**数据结构**(Data

* pre 和 post 的意思是:若前置条件在执行操作之前为真,则后置条件在执行操作后必为真。

structures)表示抽象数据类型。一个数据结构是程序中表示一个抽象数据类型的一种特定方法。一个抽象数据类型可以有不同的表示方法，对 capital 数据类型，可建立下列数据结构：

```
type
    capital=array["A"..."Z"] of boolean;
```

算法指明怎样实现数据类型的操作，它精确地规定如何使用和操纵数据类型的表示。要实际地实现一个抽象数据类型，表示和算法必须用某种程序设计语言编码。Ada 和 Modula-2 等许多程序设计语言都提供了实现抽象数据类型的设施，对 capital 类型的说明和实现，我们用 Modula-2 分别写成下列**定义模块**和**实现模块**：

```
DEFINITION MODULE capletters;
    EXPORT QUALIFIED capital, init, ismember, insert, delete;
    TYPE capital;
    procedure init (var S:capital);
    function ismember (x:char; S:capital):boolean;
    procedure insert (x:char; var S:capital);
    procedure delete (x:char; var S:capital);
END capletters.
```

```
IMPLEMENTATION MODULE capletters;
    TYPE capital=array["A"..."Z"] of boolean;
    procedure init (var S:capital);
    var c:char;
    begin
        for c:="A" to "Z" do
            S[c]:=false;
    end;
    function ismember (x:char; s:capital):boolean;
    begin
        return S[x]
    end;
    procedure insert (x:char; var S:capital);
    begin
        S[x]:=true
    end;
    procedure delete (x:char; var S:capital);
    begin
        S[x]:=false
    end
END capletters.
```

定义模块中首先是接口部分说明，在我们的例子中，接口部分只有一个受限移出表：

EXPORT QUALIFIED capital, init, ismember, insert, delete;

表中列出可供外部模块使用的每一个名，接着是对每个名的详细说明，以便其它模块使用它们。TYPE capital 把描述类型 capital 的细节完全隐蔽起来，这种数据类型为不透明的或私有的数据类型，使用该模块的用户无法知道它的结构。关于数据类型的表示和各种操作过程的实现细节放在实现模块中。对应用程序而言，实现模块中的内容是不可见的，应用程序只能通过定义模块中说明的操作来使用类型，从而保证了数据对象的安全性。当采用其它数据结构来实现抽象数据类型时，只需修改相应的操作过程，应用程序可以不变。

下面是使用上述两个模块的例子。

例 1.1 读入一个仅由大写字母组成的字符序列，序列以“$”为结束符号。统计序列中各个不同字母出现的次数以及每个字母在序列中首次出现时的序号。

为了确定当前读入的字母是否为首次出现，我们建立一个由 26 个英文大写字母组成的集合 S。每读入一个字母，便检测该字母是否属于 S，若属于 S，则为首次出现，接着从集合 S 中删除该字母。若读入的字母不属于 S，则为非首次出现。下面是解决这一问题的程序模块。

```
MODULE Countoccur;
    FROM InOut IMPORT read, writeint, writestring, writeln;
    FROM capletters IMPORT capital, init, ismember, insert, delete;
    type
        letters=["A"..."Z"];
    var
        ch:char;
        count:integer;
        S:capital;
        order:ARRAY letters of integer;
        occur:ARRAY letters of integer;
    begin
        (*建立一个空集*)
        init(S);
        for ch:="A" to "Z"do (*建立集合 S,数组 occur 和 order 初始化*)
          begin
            insert (ch,S);
            order [ch]:=0;
            occur [ch]:=0;
          end;
        count:=0;
        writestring ("please enter a character");
        read (ch);
        while ch <> "$"do
          begin
```

```
            count:=count+1;
            if ismember (ch, S) then
              begin
                delete (ch,S);
                order [ch]:=count;
              end;
            occur [ch]:=occur [ch]+1;
            writestring ("please enter a character");
            read (ch)
        end;
    writestring ("letter first occurence occur number");
    for ch:="A" to "Z" do
        begin
          writeln;
          writestring(ch);
          writeint (order [ch],occur [ch]);
        end
end Countoccur.
```

应用模块通过移入表即可使用定义模块和实现模块中的说明和过程，大大地简化了程序设计。

对标准 PASCAL 语言，一个 PASCAL 程序的说明部分应严格地按照标号说明、常量定义、类型定义、变量和子程序说明这样一种特定的顺序出现，用来建立一个抽象数据类型的成分必须被分散在其它的说明和子程序中，不能从中分离出来封装在一起组成一个独立的整体。使用某一数据类型的程序必须包含该数据类型的说明及实现过程，不能将使用和实现分开，这样就使得某些对象是全程可见的，因此不能保证安全性。尽管如此，我们仍然可用抽象数据类型的方法进行程序设计：在编程时，先写出表示一个抽象数据类型的数据结构以及实现它的各种操作的过程或函数，然后依据操纵这些抽象数据类型的操作来组织程序。

1.2 基本数据结构

本节介绍数组、记录和指针这三种基本数据结构，它们是组成复杂数据结构的基础。

数组

几乎所有的高级语言都提供数组结构，一个一维数组是有限个元素的一个序列，数组中的所有元素都具有相同的类型并占有相同数量的存储空间。要选择数组的一个元素，只需指明它在序列中的位置的整数下标即可。一个数组的说明通常包括数组名、元素的类型和下标的值的范围，在 PASCAL 中，一个数组说明如下：

```
var
```

A:array [1..10] of integer;

它指明 A 是一个含 10 个整数的数组，允许的下标值从 1 到 10。一个数组中的元素总是被顺序地存储在相邻的存储位置上。数组 A 的 10 个元素 A[1]、A[2]、…、A[10]在内存中的排列如图 1.2 所示。设每个元素在内存占 c 个字节，符号 Loc(A[i])表示第 i 个元素 A[i]的存储地址，则有

$$Loc(A[i])=Loc(A[1])+(i-1)*c$$

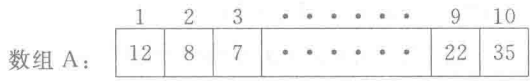


图 1.2 数组元素在内存的排列

记录

正如数组是多个(具有相同类型的)元素的序列一样，**记录**(record)也是由若干元素组成的序列，但这些元素的类型和长度可以不同。描述一个学生情况的学号(num)、姓名(name)、性别(sex)、年龄(age)、专业(speciality)五个数据项组成一个记录：

num	name	sex	age	speciality

用 PASCAL 语言描述记录的组成如下：

```
type
    element=record
              num:integer;
              name:packed array [1..20] of char;
              age:integer;
              sex:(Male, Female);
              speciality: packed array [1..20] of char
            end;
```

像数组一样，在计算机中，一个记录的诸元素也是被存储在相邻的存储单元中。

记录常被用来组合成数组，称为记录数组，例如，n 个学生记录组成数组 A，说明如下：

```
var
    A:array [1..n] of element
```

记录中的数据项称为**字段**(field)，字段又可以是一个记录或数组(如字段 name 和 speciality)。在传统的数据处理中，记录是外部文件和程序通讯的单位。

在记录上经常使用的操作是字段选择，即取出记录中某个指定字段的值或赋给该字段一个新值。选择的方法是在记录变量名与字段名之间用“.”隔开，例如 A[3].age 表示数组 A 第三个元素的 age 字段。

数组和记录这两种数据结构将作为存储结构来构造更复杂的数据结构。

指针

指针(pointer)又称**链接**，是计算机内存单元地址的抽象。一个指针被定义为一个变量，它

给出另一类型的变量的位置或地址。我们把计算机内存RAM看作是字节的一维数组，若内存所含的字节个数为m，则一个指针变量可取的值为0到m－1。如果一个指针变量不指向任何对象，则称为空指针，在PASCAL中表示为nil。图1.3中的p、q、r和s都是指针，其中p指向一个记录，q和r都指向实型变量x，s为空指针。

指针的操作有五种：：＝(赋值)、＝(相等)、<>(不等)及new和dispose。

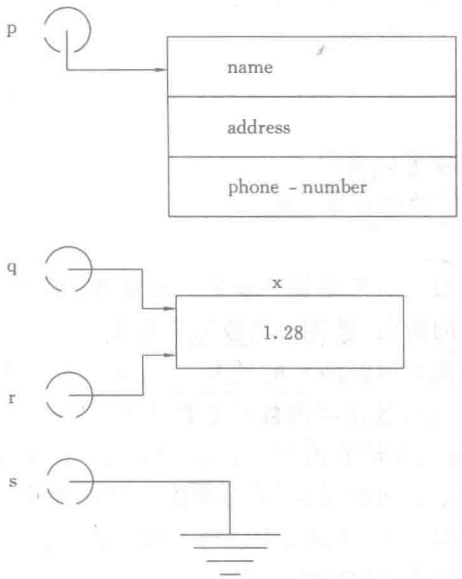


图1.3　指针示意图

1.3　算法及算法分析

算法(algorithm)是由若干条指令组成的有限序列，它精确地定义一系列规则，指出怎样从给定的输入信息得到要求的输出信息。算法中每条指令都必须是明确的，无二义性；每条指令的执行次数以及每次执行的时间都是有限的。算法和程序的区别是程序的执行可以是无限的，例如操作系统程序，只要系统不受破坏，操作系统的运行将不会停止。一个程序是一个算法，只要该程序对任何输入，不会发生死循环。

我们来看一个算法的例子。设有n个整数$a_1,a_2,\cdots,a_{n-1},a_n$，要将它们按递增的顺序排列。采用的方法是将相邻两个数a_{j-1}和a_j($1<j\leqslant n$)比较，若$a_j<a_{j-1}$，则交换a_j和a_{j-1}，否则不作交换。算法先比较a_{n-1}与a_n，然后再看a_{n-2}与a_{n-1}，依此类推，直到a_1与a_2。经过从右到左一趟处理后，最小的数将处在第一个位置上。然后再用同样的方法进行第二趟处理，使其余n－1个数中的最小者处在第二个位置上，……。显然经过n－1趟处理后将完成排序。

设待排序的n个数组成数组A，上述排序算法用PASCAL语言描述如下：

```
procedure bubble-sort (var A[ 1..n] of integer);
    var i,j,temp:integer;
    begin
```

```
①      for i:=1 to n-1 do
②          for j:=n downto i+1 do
③              if A[j-1]>A [j] then begin
                   {交换 A[j-1]与 A[j]}
④                  temp:=A[j-1];
⑤                  A[j-1]:=A[j];
⑥                  A[j]:=temp
               end
       end.
```

评价一个算法的性能主要考虑两点：

(1) 算法的执行时间和需要的存储空间。

(2) 程序设计的工作量。

一个程序的运行时间主要与程序的输入量的大小有关，输入量的大小反映了被处理问题的规模，例如，两个相乘矩阵的阶数，被排序的数的个数等等。设输入量的大小用 n 表示，程序的运行时间用 T(n)来表示，我们讨论随 n 的增大 T(n)的变化情况。

在排序程序 bubble-sort 中，被排序的数的个数为 n，排序过程中的基本操作是数据的比较和交换(即程序中的条件测试和赋值语句)，T(n)的大小与这些语句被执行的次数有关。语句④、⑤、⑥是否被执行依赖于比较的结果，假定每次比较都要执行这三个语句，内层 for 循环执行(n−i)次，外层 for 循环执行 n−1 次。设进行一次比较和执行三次赋值的时间为 c，于是，可以计算在最坏情况下程序的执行时间为：

$$T(n)=\sum_{i=1}^{n-1}c*(n-i)=c*\sum_{i=1}^{n-1}(n-i)=c*\frac{n(n-1)}{2}=\frac{c}{2}n^2-\frac{c}{2}n$$

略去 n 平方项的系数及 n 的一次项，将上式写成 $T(n)=O(n^2)$，其含义是随着 n 的增大，T(n)的增长速度小于 n^2 的增长速度。

更确切地说，如果一个算法的执行时间 T(n)是 O(f(n))(T(n)与 f(n)是定义在自然数域 N 上的函数)，则一定存在两个正常数 c 和 n_0，使对于所有的 $n\geqslant n_0$，$T(n)\leqslant cf(n)$成立。一般说，T(n)与 f(n)是同阶的，O(f(n))通常称为该算法的**时间复杂度**(Time complexity)。

用 O(f(n))表示算法的执行时间时，应使 f(n)尽可能简单。若一个算法的时间 $T(n)=3n^3+2n^2$，则说 T(n)是 $O(n^3)$。因为可取 $n_0=0$，$c=5$，当 $n\geqslant n_0$ 时，$3n^3+2n^2\leqslant 5n^3$。也可以说，T(n)为 $O(n^4)$，但用 $O(n^3)$更能恰当地说明 T(n)的增长速度。

在本书后面的各章中，我们将会遇到表示时间复杂度的常用符号。O(1)表示计算时间是一个常数，不依赖于 n；O(n)表示时间与 n 成正比或称是线性的；$O(n^2)$、$O(n^3)$、$O(2^n)$分别称为平方阶、3 阶和指数阶的；$O(\log_2 n)$和 $O(n\log_2 n)$为对数阶的。这 7 个函数随着 n 的增大而变化的情况，如图 1.4 所示。

某些算法的执行时间不仅与输入量的大小有关，还可能与输入的具体数据有关。也就是说，当输入量的大小 n 固定时，执行基本操作的次数可能取决于特定的输入。在这种情况下，应当计算算法的平均时间，即对输入量大小为 n 的所有不同输入，求它们的执行时间的平均值。要精确地计算平均时间，必须知道不同输入出现的概率。但这些概率值一般不能解析地计算，所以平均时间的计算有时是相当困难的。在这种情况下，我们定义对大小为 n 的任何输入，

T(n)为运行时间的最大值。当说到一个算法的执行时间时，除非特别说明，一般总是指最坏情况下的时间。另外，程序的运行时间还与编译程序生成的目标的质量以及执行程序所用的机器的性能和速度等因素有关。为了简化讨论，本书中出现的算法都假定是在一台假想的虚拟机上运行，不考虑这些因素。

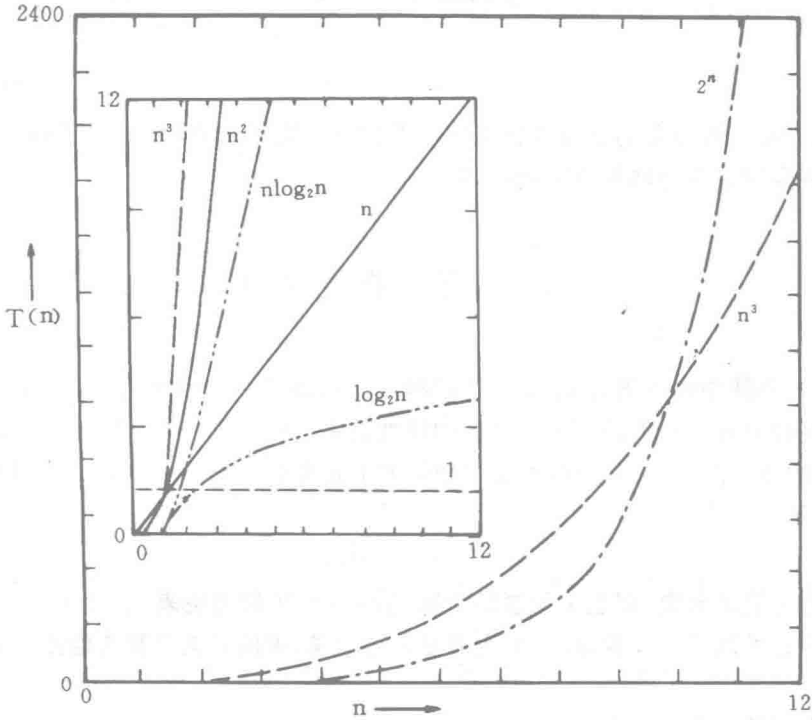


图 1.4　常用函数的增长率

一个程序所需的空间包括存储程序本身的指令、变量、常数和输入数据的空间，此外还需要一些额外的空间对数据进行操作。如果输入数据有自己自然的形式，例如一个数组或矩阵，则仅需考虑额外存储空间的大小与输入量的关系。如果输入数据可以有不同的表示形式，则应同时考虑输入数据本身占用的空间量与额外的空间量，其计算方法与计算执行时间的方法相同。

如果一个算法很难进行程序设计，即使它有较高的效率，有时也是不可取的。因为程序员编程和调试花费的代价可能大大超过程序运行的代价，特别是在程序只执行一次的情况下，我们宁愿选择一个效率较低但较简单的算法，因为它可能使我们容易验证算法的正确性，容易调试和改进程序。如果这个程序要使用多次，则效率可能是选择算法的决定因素，这时应选择一个较快的算法，尽管这种算法比较复杂。

第二章　表

本章主要讨论一种重要的具有线性有序关系的抽象数据类型——表(LIST),以及实现表的各种方法,并介绍表在存储管理中的应用。

2.1　抽象数据类型表

一本电话号码簿中的各项,售票窗口等候购票的顾客,图书馆书架上的图书,……,所有这些都有一个共同的特征:它们都有一个自然的线性次序,具有这一特征的一组元素统称为"**表**"(list)。就数学的意义讲,一个表是由给定类型的若干元素组成的一个序列。通常用括在括号中的一列元素表示一个表:

$$(a_1,a_2,\cdots,a_i,\cdots,a_n)$$

这里,n(≥0)称为表的**长度**,即表中元素的个数。若 n=0,则称为**空表**。表中每个元素都取自某个对象的集合,它可以是一个数,或一个大而复杂的记录,表的元素又称为**结点**(node)。下面是一些表的例子。

由 26 个大写英文字母组成的表:

(A,B,C,D,…,X,Y,Z)

一副纸牌的点数组成一个表:

(2,3,4,5,6,7,8,9,10,Jack, Queen, King,Ace)

下列一份学生情况登记表可看作一个表,它的每个元素是描述一个学生情况的记录。

学　号	姓　名	年　龄	专　业
863201	王　明	18	数　学
863202	张　红	19	数　学
863203	李　平	18	数　学
863204	陈　博	17	计算数学
863205	赵　瑞	20	计算数学
863206	吴　华	17	数　学
863207	周　苏	19	数　学
863208	郑　康	18	计算数学

表的一个重要特征是可以按照诸元素在表中的位置确定它们在表中的先后次序。假定 n≥1,元素 a_i 处于表的第 i 个位置上,a_i 先于 a_{i+1}($i=1,2,\cdots,n-1$),a_{i+1}称为 a_i 的**直接后继**