严蔚敏 吴伟民 编著

数据结构

SHUJU JIEGOU

清华大学出版社

数 据 结 构

严蔚敏　吴伟民　编著

清华大学出版社

内 容 简 介

本书系统地介绍了各种类型的数据结构和查找、排序的各种方法。对每一种数据结构，除了详细阐述其基本概念和具体实现外，并尽可能对每种运算给出类 PASCAL 的算法，对查找和排序的每个方法，除了给出相应的算法外，并着重在时间上作定量或定性的分析比较。在最后一章还讨论了文件的各种组织方法。

本书概念清楚，内容丰富，并配套了一定量的习题和上机实习题，既便于教学，又便于自学。

本书可作为大专院校计算机系和计算机应用专业的教材，也可供从事计算机工程与科学工作的科技工作者参考。

数 据 结 构

严蔚敏　吴伟民　编著

☆

清华大学出版社出版发行

北京　清华园

清华大学印刷厂印刷

☆

开本：787×1092 1/16　印张：23.75　字数：592 千字

1987 年 1 月第 1 版　　1987 年 1 月第 1 次印刷

印数：00001—30000

统一书号：15235·207　　定价：3.90 元

前　言

本书详细介绍了线性表、栈和队列、串、数组和广义表，树和二叉树以及图等几种基本类型的数据结构，以及在程序设计中经常遇到的查找和排序问题。全书共分十二章，在第一章中首先从三个实际问题引出的非数值性程序设计例子出发通俗地回答了什么是数据结构，并综述了数据、数据类型及数据结构等基本概念，然后对书中描述算法所用的语言及算法分析作了粗略说明；在第二章至第七章中分别介绍了上述几种数据结构，对每一种结构力求从数据元素之间存在的关系出发给出恰当的定义，为了明确表示数据结构在计算机中的表示，本书中采用了类似于 Pascal 语言的类型说明来定义存储结构，并在讨论基本运算的基础上给出一些应用例子。第八章综合介绍操作系统和编译程序中涉及的动态存储管理的基本技术，第九章至第十一章讨论查找和排序两个重要问题，这里，除了给出各种方法的算法之外，还着重从时间上对这些方法作定性或定量的分析和比较；第十二章讨论了文件的物理结构。

本书可作为计算机系本科学生的教材，讲授学时为 80～90。在学时少的情况下，讲授教师可根据本校学生的情况酌情挑选一些内容。本书在目录页中给出参考性的意见（建议删去带＊＊的章节）；对偏于计算机应用专业的学生，还可以删去第五、八和第十一章的内容。为了适应当前教改的需要，便于学生进行自学，本书在文字叙述上力求做到语言通俗、简明易懂，特别是对第二章至第七章的内容，解释颇为详细。绝大多数的运算都给出了类 PASCAL 语言描述的算法，只要稍加修改，便可变成可上机执行的 PASCAL 程序。

《数据结构》从课程性质上来讲是一门专业技术基础课。它的教学要求是：学会分析研究计算机加工的数据对象的特性，以便选择合适的数据结构和存储结构以及相应的算法，初步掌握各种算法在时间和空间的分析技巧。另一方面，学习本课程的过程也是进行复杂程序设计的训练过程，要求学员书写的程序结构清楚、正确易读，因此，整个教学过程中，习题和上机实习是两个不可缺少的环节，本书各章习题和实习集中在一起另编成册。

本书也可供从事计算机应用等工作的工程与科技人员参考。只需掌握编制程序的基本技术便可学习本书，若具有离散数学和概率论知识的读者则更容易些。

本书由沈阳工业学院曹素芬同志和北京计算机学院的陈文博同志审阅了部分初稿，并提出许多宝贵意见。全书由金慧芬同志帮助抄写，在此一并表示衷心感谢。

本校于 1980 年编写的教材曾得到许多读者的关切，来信指正书中错误，在此向全体读者表示敬意和感谢。由于水平所限，本书中仍难免有错误和缺点，恳切希望继续得到本书读者特别是讲授此课程的教师们的批评指正。

编者

1986 年 4 月

目　录

第一章 绪 论

自第一台计算机问世以来，计算机的飞速发展已远远超出人们对它的估计，甚至在有的生产线上几秒钟就能生产出一台微型计算机，这使得它的应用范围大大扩展，至今，计算机已深入到人类社会的各个领域。计算机已不仅用于科技计算，而更多地用于数据处理和实时控制。与此相应，计算机加工处理的对象也从纯碎的数值发展到字符、图象、声音等各种复杂的具有一定结构的数据。因此，要设计出一个“好”的程序，必须研究数据的特性以及数据之间存在的关系。这就是“数据结构”这门学科形成和发展的背景。

1.1 什么是数据结构

从提出一个实际问题到计算机解出答案需要经过下列步骤：首先从实际问题抽象出一个数学模型，然后设计一个解此数学模型的算法，最后编出程序、进行测试、调整直至得到最终解答。寻求数学模型的实质是分析问题，从中提取操作的对象以及这些操作对象之间含有的关系，然后用数学语言加以描述。例如，在分析了一个物理现象或化学现象变化的规律之后可以得到一组代数方程或微分方程。然而，更多的问题无法用数学方程加以描述。下面我们来看三个例子。

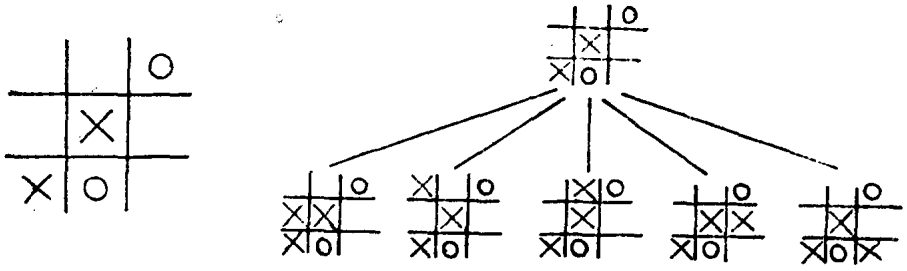
例一：计算机管理图书目录问题。当你想借阅一本参考书但不知道书库中是否有的时候；或者，当你想找某一方面的参考书而不知图书馆有哪些书的时候，你都需要到图书馆去查阅目录卡片。在图书馆内有各种明目的卡片，有按书名编排的，有按作者编排的，还有按分类编排的等等。若利用计算机帮助查询书目首先需要将这些目录卡片存入计算机，如何存放？既要考虑查询时间短，又要考虑节省空间。一个最简单的办法是建立一张表，每一本书的信息，只用一张卡片表示，在表中占一行，如图 1.1 所示。此时计算机操作的对象（数据元素）便是卡片，卡片之间的关系是顺序排列的，计算机对数据的操作是按照某个特定要求（如给定书名）进行查询，找到表中满足要求的一行信息。由此，从计算机管理图书目录的问题抽象出来的数学模型便是包含图书目录的表和对表进行查找运算。

书 名	作者名	登录号	分 类	出版年月

图 1.1 图书目录表

例二：计算机和人对弈问题。计算机之所以能和人对弈是因为有人将对弈的策略已事先存入计算机。由于对弈的过程是在一定规则下随机的，为使计算机能灵活对弈就必须将对弈过程中所有可能发生的情况以及相应的对策都考虑周全。而且，在决定对策时

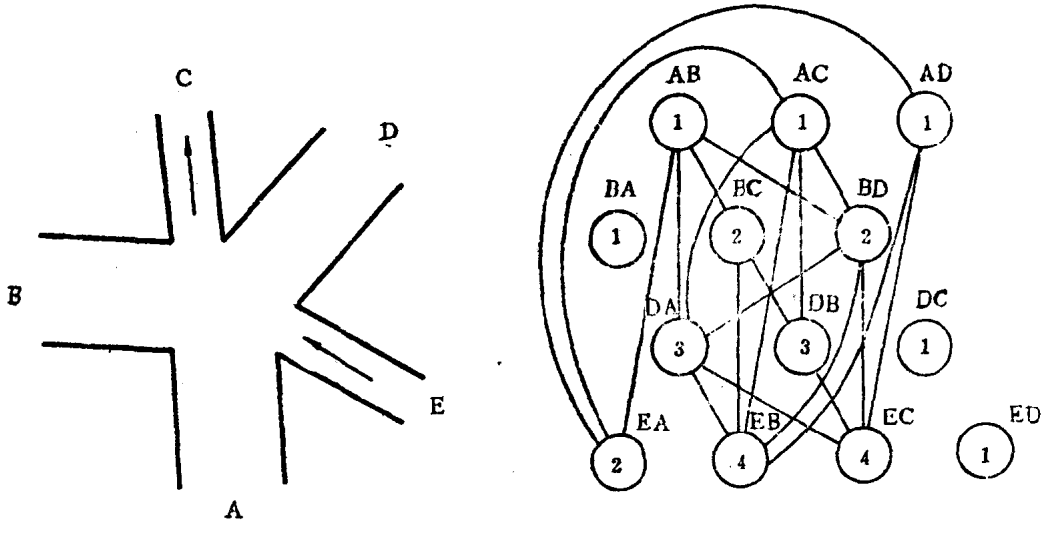
不仅要看当时的棋盘状态，还要考虑发展趋势直至最后取胜的可能性。由此，计算机操作的对象（数据元素）是对弈过程中每一步的棋盘状态（格局）。数据元素之间的关系由比赛规则决定。通常，这个关系不是线性的，因为从一个棋盘格局可以派生出几个格局。如图 1.2(a) 所示井字棋① 的一个格局，下一步由持 X 子的甲方走棋，则有五种可能出现的格局，如图 1.2(b) 所示，这个图好比由树的主叉派生出五个分叉，我们称它为树，它可用来形象表示某一类问题中数据元素之间的关系。由此对弈的问题也就转换成生成一棵对弈树并对其进行某种操作。



(a)　　(b)

图 1.2　井字棋对弈树

(a) 对弈树的数据元素　　(b) 对弈树的一部分



(a)　　(b)

图 1.3　五叉路口交通管理示意图

(a) 五叉路口　　(b) 表示通路的图

① 井字棋由两人对弈，棋盘为 3×3 的方格，当一方的三个棋子占同一行、或同一列，或同一对角线时便为赢方。

例三：多叉路口交通灯的管理问题。通常，在十字交叉的路口只要设红绿两色的交通灯便可保持正常的交通秩序，而在多叉路口需设几种颜色的交通灯才能既使车辆相互之间不碰撞而又达到最大的流通呢？假设有如图 1.3(a) 所示的五叉路，其中 C 和 E 为单行道，在路口有 13 条可行的通路，其中有的可以同时通行，如 A→B 和 E→C，而有的不可同时通行，如 E→B 和 A→D，那末，在路口应如何设置交通灯进行管理？这个问题可以转换成一个图的染色问题。假设在图上以一个圆圈表示一条通路，在不能同时通行的两个圆圈之间画一连线，对图中的圆圈上色，要求同一连线上的两个圆圈不同色且颜色种类最少。图 1.3(b) 是与图 1.3(a) 相应的图，图中 13 个圆圈表示 13 条通路，圆中的号码分别表示四种颜色的交通灯。

综上三个例子可见，描述这样一类问题的数学模型不再是数值方程，而是诸如表、树和图等的非数值性的数据结构及其运算。因此，简单说来，数据结构就是一门研究非数值性程序设计中计算机操作的对象以及它们之间的关系和运算等等的学科。

1.2 基 本 术 语

在这一节中，我们将对全书中常用到的名词和术语赋以确定的含义。

数据（data）是描述客观事物的数、字符，以及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。它是计算机程序加工的“原料”。例如，一个利用数值分析的方法解代数方程的程序处理的对象只是整数和实数，而一个编译程序或文字处理程序的对象是字符串。因此，对计算机而言，数据的含义极为广泛，如图形、声音等都属于数据的范畴。

数据元素（data element）是数据的基本单位，即数据这个集合中的一个个体（客体）。有时一个数据元素可由若干个**数据项**（data item）组成，数据项是数据的最小单位。例如，图 1.1 中每一行（即一本书的信息）为一个数据元素，而其中的每一项（如书名、作者名等）为数据项。

数据对象（data object）是具有相同特性的数据元素的集合，是数据的一个子集。例如，整数的数据对象是集合 $N=\{0,\pm1,\pm2,\cdots\}$，字母字符的数据对象是集合 $C=\{A,B,\cdots,Z\}$。

数据结构（data structure）简单说来，数据结构是带有结构的数据元素的集合。从上述三个例子可以看到，被计算机加工的数据元素（如：一本书的信息，棋盘的一个格局，一条可行的通路等）都不是孤立的，在它们之间存在着某种连系，这种相互之间的关系，通常称做结构。我们可以从集合论的观点加以形式化描述。

数据结构是一个二元组

$$\text{Data-Structure}=(D,R)$$

其中：D 是数据元素的集合，R是 D 上关系的集合。例如，复数可被定义为一种数据结构

$$\text{Complex}=(D,R)$$

其中 $D=\{x|x$ 是实数$\}$

$R=\{R_1\}$

$R_1=\{\langle x,y\rangle|x,y\in D,$ x 称为实部，y 称为虚部$\}$

由此，由任何一对实数均可得到一个复数。又如，整数、实数也是数据结构。由于它们的数据元素之间的关系都很简单，我们称它们为初等数据结构。

在上面对数据结构的定义中，R 集合中的关系指的是数据元素之间的逻辑关系，因此，又称数据的逻辑结构。与此对应，则有

数据的物理结构，又称存贮结构，是数据结构在计算机中的映象（或表示）。它包括数据元素的映象和关系的映象。在计算机上处理信息的最小单位是一位二进制数，叫做位（bit），在计算机中，我们可以用一个由若干位组合起来形成的一个位串来表示一个数据元素，称这个位串为元素（element）或结点（node）。当数据元素由若干个数据项组成时，则位串中对应于每个数据项的子位串称做数据域（data field）。由此，元素或结点是数据元素在计算机中的映象或表示。由于映象的方法不同，数据元素之间的关系在计算机中有两种不同的表示：顺序映象和非顺序映象，并由此得到两种不同的存贮结构：顺序存贮结构和非顺序存贮结构（又称链式存贮结构）。任何一个算法在编制程序之前首先需选择合适的存贮结构，另一方面，同一运算在不同的存贮结构中实现的方法不同，则有的问题的算法设计依赖于所选取的存贮结构。因此，对计算机的程序设计而言，数据的逻辑结构和物理结构是紧密相连的两个方面，因此，在有些数据结构的书中并不严格区分之。

数据类型（data type）是程序设计语言中所允许的变量的种类，换句话说，是变量可能取的值和能作的运算的集合。在每一种程序设计语言中都有一组它所允许的基本数据类型。如 FORTRAN 语言提供了五种基本数据类型：整数、实数、双精度数、复数和布尔量，它不仅规定了每一种类型变量的取值范围，还规定了该类型变量能执行的运算。又如 Pascal 语言中不仅提供了整数等四种标准类型，还提供了其它一些类型。各种语言所能提供变量类型的多少决定了该语言功能的强弱。我们可以把数据类型看作是程序设计语言中已经实现的数据结构，如复数、数组等。因此，数据类型实际上是数据结构（包括逻辑结构和存贮结构）及其运算的总称。在本书中将以类型说明形式化地描述存贮结构。

1.3 数据结构的发展简史及它在计算机科学中所处的地位

《数据结构》作为一门独立的课程在国外是从 1968 年才开始的。在这之前，它的某些内容曾在其它课程，如表处理语言中所讲述。1968 年，在美国一些大学的计算机系的教学计划中，虽然把《数据结构》规定为一门课程，但对课程的范围仍没有作明确规定。当时，数据结构几乎和图论，特别和表、树的理论是同义语。随后，数据结构这个概念被扩充到包括网络、集合代数论、格、关系等方面，从而变成了现在称之为《离散结构》的内容。然而，由于数据必须在计算机中进行处理，因此，不仅考虑数据本身

的数学性质，而且还必须考虑数据的存贮结构，这就进一步扩大了数据结构的内容。近年来，随着数据库系统的不断发展，在数据结构课程中又增加了文件管理（特别是大型文件的组织等）的内容。

1968 年出版的美国唐·欧·克努特教授所著《计算机程序设计技巧》第一卷《基本算法》是第一本较系统地阐述数据的逻辑结构和存贮结构及其运算的著作，从六十年代末到七十年代初，出现了大型程序，软件相对独立，结构程序设计成为程序设计方法学的主要内容，人们越来越重视数据结构，认为程序设计的实质是对确定的问题选择一种好的结构加之设计一种好的算法。从七十年代中期到八十年代初，各种版本的数据结构著作相继而出。

目前在我国，《数据结构》已经不仅仅是计算机专业的教学计划中的核心课程之一，而且是其它非计算机专业的主要选修课程之一。

《数据结构》在计算机科学中是一门综合性的专业基础课[3]。数据结构的研究不仅涉及到计算机硬件（特别是编码理论、存贮装置和存取方法等）的研究范围，而且和计算机软件的研究有着更密切的关系，无论是编译程序还是操作系统，都涉及到数据元素在存贮器中的分配问题。在研究信息检索时也必须考虑如何组织数据，以便查找和存取数据元素更为方便。因此，可以认为数据结构是介于数学、计算机硬件和计算机软件三者之间的一门核心课程（如图 1.4 所示）。在计算机科学中，数据结构不仅是一般程序设计（特别是非数值性程序设计）的基础，而且是设计和实现编译程序、操作系统、数据库系统及其它系统程序和大型应用程序的重要基础。

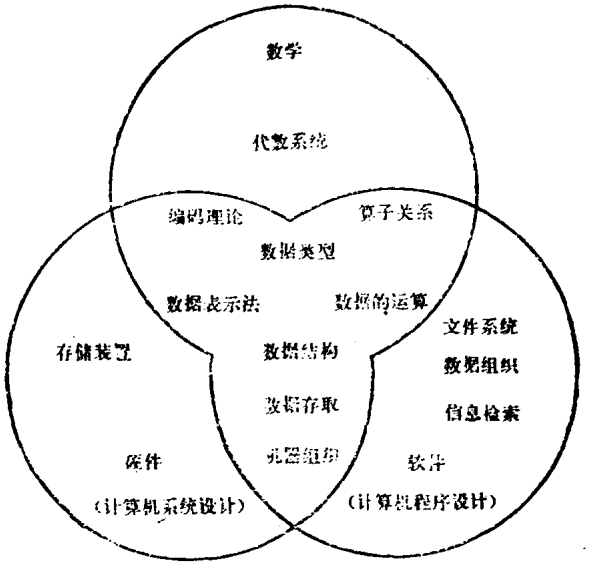


图 1.4 《数据结构》所处的地位

1.4 关于算法的描述和算法分析的说明

由于研究数据结构的目的在于更好地进行程序设计，因此，本书在讨论各种数据结构的基本运算时都需要给出程序。但是，由于用某种程序设计语言书写一个正规的程序会带来很多不便（如繁琐的变量说明，某些语句上的限制使程序不能一目了然等等），因此，本书采用类 Pascal 语言（类似于 Pascal，但又不同于 Pascal）作为一种假想的教学语言来描述各种运算，为和“程序”区别之，在此借用“算法”① 一词代之。

① 关于“算法”，克努特教授曾给出确切的定义，在此不作仔细探讨。读者可参阅《程序设计技巧》第一卷中译本 pp.3。

现对本书中所用的类 Pascal 语言作如下说明：

(1) 所有的算法都以如下过程或函数的形式表示

```
PROCEDURE 过程名（参量表）；        FUNCTION 函数名（参量表）：类型名；
  BEGIN                               BEGIN
    语句组                              语句组
  END；{过程名}                       END；{函数名}
```

其中，参量表可以含有若干值参和变参；语句组由一或一个以上的语句组成，用“；”作为两个语句之间的分隔符。在正常情况下，过程结束于 END；非正常出口语句 RETURN 表示跳出过程。在函数过程中，函数值 f 以 RETURN(f) 形式返回调用该函数的过程。

(2) 除过程中的参量表外，所有在过程中出现的局部变量均不加以变量类型的形式说明（个别情况例外）。并设想该语言允许一切可能出现的变量种类。

(3) 基本语句有

赋值语句

```
变量名:=表达式;
```

条件语句

```
IF 条件 THEN 语句1；
```

或

```
IF 条件 THEN 语句1 ELSE 语句2；
```

其中：语句 1 和语句 2 都可以是任意的语句组。当语句组包含多于一个语句时，则用方括弧将这些语句括起来。以下说明中出现的“语句”均与类同。

循环语句

```
WHILE 条件 DO 语句;
```

或

```
REPEAT 语句组 UNTIL 条件;
```

或

```
FOR 变量:=初值 TO 终值 DO 语句;
```

或

```
FOR 变量:=初值 DOWNTO 终值 DO 语句;
```

在后两个循环语句中，前者的终值大于或等于初值，增量为 1；后者的终值小于或等于初值、增量为 −1。

情况语句

```
CASE
  条件1：语句1；
  条件2：语句2；
    ...
```

条件 n:　　语句 n;
(ELSE　　语句 n+1)
END;

或

CASE 变量名 OF
值 1：语句 1
⋮
值 n：语句 n
END;

调用过程语句

CALL 过程名（参量表）；

过程允许嵌套调用和递归调用。

函数调用语句

变量名:=函数名（参量表）；

出错处理语句

ERROR（字符串）

跳出循环语句

EXIT

(4) 输入和输出　假定存在两个标准过程

read（变量表）；　　write（变量表）；

其中变量表可由若干变量名或字符串组成，中间用逗号隔开。

(5) 在过程中的任何处均可插入用一对花括弧括起来的中文注解。

(6) 其它

- 在参量表中出现的类型，其说明在讨论存贮结构时进行；
- 类型名、过程名、变量名的字符长度不限；
- 在引用各种数据结构的基本运算和标准过程时，前面不加保留字 CALL，而直接写运算名或标准过程名。

前面已经提及，在编制程序之前，首先应选择恰当的数据结构和一个好的算法。那末，如何衡量一个算法的好坏呢？

显然，选用的算法应是“正确的”①。除此之外，通常有三个方面的考虑：

(1) 依据算法编制成程序后在计算机中运行时所消耗的时间；

(2) 依据算法编制成程序后在计算机中所占存储量的大小，其中主要考虑程序运行时所需辅助存储量的大小。

(3) 其它诸如：算法是否易读、是否容易转换成任何其它可运行的语言编制的程序以及是否易被测试等等。

① 对算法的正确性的验证非本课程研究的课题，故在此不作详细讨论。

从主观上讲，我们希望选用一个既不占很多存储、运行时间又短，其它性能也好的算法。然而，实际上不可能做到十全十美。往往是，一个看来很简便的程序，其运行时间要比一个形式上复杂的程序慢得多；而一个运行时间较短的程序往往占用的辅助存储量较大。因此，在不同情况下应有不同的选择。若该程序只使用一次或几次，则力求算法简明易读，易于转换成上机的程序；若该程序需反复运行多次，则应选用运行时间尽可能少的算法；又若待解问题的数据量甚大，而所用的微机存储空间较小，则相应算法应主要考虑如何节省空间。在本书的讨论中，将主要讨论算法的时间特性，偶尔也讨论其空间特性。

一个程序在计算机上运行时所需耗费的时间取决于下列因素：

（1）程序运行时所需输入的数据总量；

（2）对源程序进行编译所需时间；

（3）计算机执行每条指令所需时间；

（4）程序中的指令重复执行的次数。

前三条依赖于实现算法的计算机软、硬件系统。如果计算机可以进行脱机输入，则输入数据的时间可以忽略不计。其中（2）和（3）依赖于实现算法所用语言的编译程序的性能和计算机本身的速度。因此，习惯上常常把语句重复执行的次数作为算法的时间量度。

在此，需引入一个语句的频度（frequency count）的概念。语句的频度即为语句重复执行的次数。例如，两个 $n\times n$ 的矩阵相乘，其算法可描述如下：

```
FOR  i:=1  TO  n  DO                                    n+1
  FOR  j:=1  TO  n  DO                                  n(n+1)
    [c[i, j]:=0;                                        n²
      FOR  k:=1  TO  n  DO                              n²(n+1)
        c[i,j]:=c[i,j]+a[i,k]*b[k,j]]                   n³
```

其中每一语句的频度如上右列所示，整个程序中所有语句的频度之和可约定作为该程序运行的时间量度，记作 $T(n)=2n^3+3n^2+2n+1$。显然，它是矩阵的阶 n 的函数，并且，当 $n\to\infty$ 时，$T(n)/n^3\to 2$。若引入“O”记号（读作“大O”），则有 $T(n)=O(n^3)$，意为，在 n 较大时，该程序的运行时间和 n^3 成正比，或者说，$T(n)$ 数量级和 n^3 的数量级相同，称 $T(n)$ 为算法的时间复杂度（time complexity）。

一般情况下，n 为问题的规模（大小）的量度，如矩阵的阶，多项式的项数，图中的顶点数等等。一个算法的时间复杂度 $T(n)=O(f(n))$①。因此，通常可以通过判定程序段中重复执行次数最多的语句的频度来估算算法的时间复杂度。例如，下列三个简单的程序段：

(a) x:=x+1;

(b) FOR i:=1 TO n DO x:=x+1;

① “O”形式定义[2]为：若 f(n)是正整数 n 的一个函数，则 $x_n=O(f(n))$ 表示存在一个正的常数 M，使得当 $n\geqslant n_0$ 时都满足 $|x_n|\leqslant M|f(n)|$。至于 M 和 n_0，我们不必，也无法说出它们的确切数值。