# 算法
# 和数据结构手册

G·H·戈内特 著 张子让 周晓东 译 周伯生 校



人民邮电出版社

# 算法和数据结构手册

G. H. 戈内特 著

张子让 周晓东 译

周伯生 校

人 民 邮 电 出 版 社

# 算法和数据结构手册

G. H. 戈内特 著
张子让 周晓东 译
周伯生 校

*

人民邮电出版社出版
北京东长安街27号

广益印刷厂印刷
新华书店北京发行所发行
各地新华书店经售

*

开本：850×1168 1/32 1988年5月 第一版
印张：10 16/32 页数:168 1988年5月北京第1次印刷
字数：275千字 印数：1 — 8 000 册
ISBN7-115-03544-X/TP·006
定价：3.25元

# 前　言

计算机科学就其整个发展过程而言，与其说是一门科学不如说是一门艺术。对于这一点我最喜欢举的例子就是将一个大型软件项目（例如写一个编译程序）与任何其他大型工程（例如多伦多市 CN 塔的建造）相比较。假如在审查设计的过程中，竟然将这个塔推倒重建几次，更坏的情况是，该塔已经对公众开放，然后才发现它还存在着致命的缺陷，那简直是不可想象的。然而，在软件生产中几乎人人都在使用着这种工作方法。

“站在前人的肩膀上”目前是非常困难的，多半时间我们至多只是站在前人的脚上。这本手册是为计算机科学家、教员或程序员服务，使他们能接触到这个领域在过去二十年里积累起来的知识财富。

本书中大部分结果选自所列的参考书。作者对有些结果进行了完善和推广。准确无误自然是我们的一个目标。因此，热烈欢迎读者对本书中出现的任何类型的错误提出批评和改正意见。

C. H. 戈内特

# 目　录

# 第 1 章
# 绪　言

这本手册力求包含关于算法及其数据结构的尽可能多的知识，使它能为广泛的读者服务——从希望编写高效率代码的程序员，到需要快速查阅资料的学生或研究人员。

本书重点放在算法上，给出算法的描述、用一种或几种语言编写的代码、理论上的结论以及大量的参考资料。

## 1.1 章节结构

本手册按论题划分章节。第 2 章给出了算法和数据结构的形式化描述，第 3 至第 6 章分别讨论检索、排序、选择和算术算法。附录 I 给出数据处理中常见的一些概率分布；附录 II 包含与算法分析有关的若干渐近公式；附录 III 列出重要的参考书目，附录 IV 是某些算法的另一种编码。

叙述算法的各章按需要再分为节和小节。每一个算法在小节中叙述，格式大致相同；但当资料不全或过于琐细时，我们也做了一些细微的改动或省略。一般的格式为：

(1) 算法的定义和说明，并依据第 2 章所描述的基本运算对该算法加以分类。

(2) 算法复杂度的理论结果。我们主要对那些能指出算法的运行时间及空间需求的度量感兴趣。其算法可能要进行的比较、数据存取、赋值以及变换等的次数都是度量运行时间的有用的量。算法

的某一实现中所用的词、记录、或指针等的数目是考察空间需求时要考虑的量。时间复杂性则涉及到更加广泛的度量。例如，在研究检索算法时，当我们对一个结构进行

$$\left\{\begin{array}{l}\text{查询}\\\text{添加一个记录}\\\text{删除一个记录}\\\text{修改一个记录}\\\text{重组}\\\text{建立}\\\text{顺序阅读}\end{array}\right\}$$

时，我们可以对所需要的

$$\left\{\begin{array}{l}\text{比较}\\\text{存取}\\\text{赋值}\\\text{交换}\\\text{函数调用}\end{array}\right\}\text{的}\left\{\begin{array}{l}\text{平均}\\\text{方差}\\\text{最小}\\\text{最坏情况}\\\text{平均最坏情况}\end{array}\right\}\text{数}$$

中的大部分组合作出有意义的解释。我们也可能提供其他一些理论结果，例如枚举、函数生成、或当数据元素遵循某一特定分布时的算法性能。

(3) 算法。我们选择Pascal和C作为描述算法的语言。对于那些在实践中会用到的算法，我们给出了用这两种语言中的一种或两种所编写的代码。对于那些只有理论意义的算法，则不提供代码。既用Pascal编码、又用C编码的算法，则其中一个在正文中与算法一起给出，另一个放在附录IV中。

(4) 评注。在算法描述之后，提出如何使用该算法的几点建议。我们指出编码时的注意事项；建议在什么情况下使用该算法，在什么情况下不该使用；告诉用户可以期望该算法得到最佳性能和最坏性能的条件；以及其他种种评注。

(5) 表格。只要有可能，我们都将给出表格，显示在所选用例

子中复杂性度量的准确值，使读者对算法的性能有一个感性认识。当还没有可用的精确的理论结果时，我们则给出模拟结果。模拟结果通常是以 $xxx \pm yy$ 的形式表示；这里，是根据使结果具有 95% 的置信度的要求来选择 $yy$ 的值的。换句话说，复杂性度量的值在每 20 次模拟中只有一次不在结果区间里。

(6) 内存与外存的区别。有些算法在内存比在外存有更好的性能，另一些则反之。遇到这种情况，我们将对这两种不同场合的应用给出评注。鉴于在这一问题上的大部份讨论都隐含着假定使用的是内存，在这一节中我们对使用外存的例子进行较为仔细的考察（如果合适的话）。我们将分析在使用外存时算法的性能，并讨论在实际使用时着重要考虑的问题。

(7) 在描述算法的同时，我们还给出有关的参考资料。一般的参考资料、综述或辅导材料列在章节的末尾，附录 III.2 按作者姓氏字母顺序列出所有的参考资料以及与有关算法的交叉引用关系。

## 1.2 变量的命名

贯穿这本手册，变量的命名既考虑到符号的一致性，又兼顾到在某一特殊领域里公认的术语。

除去极少数明确说明的例子以外，我们假定：

$n$ 表示一个结构中目标、元素或成分的个数；

$m$ 表示结构的大小；

$b$ 表示桶的大小，或一个物理块中元素的最大个数；

$d$ 表示数字的基数或字母表的大小。

整本手册中复杂性的度量的命名也采取了前后一致的方法。$X_n^Z$ 表示复杂性的度量，应读做“在大小为 $n$ 的结构上执行 $Z$ 时所进行的 $X$ 的次数。”

典型的 $X$ 值是：

$A$：存取、探测或结点审查；

$C$：比较或结点审查；

$E$：外部存取；

$h$：递归结构的高度（树是典型的例子）；

$I$：迭代（或函数调用次数）；

$L$：（路径或最长探测序列的）长度；

$M$：移位或赋值（通常与记录或键码的移动有关）。

典型的Z值为：

null（无上标）：成功的搜索（或当只有一种可能性时缺省操作）；

$C$：结构的构造（建立）；

$D$：一个元素的删除；

$E$：一个元素的抽取（大多数情况是为了优先级排队）；

$I$：一个新元素的插入；

$M$：若干结构的合并；

Opt：优化构造或优化结构（这个操作通常为隐含的）；

MM：最坏情况下$X$的极小化极大或极小数，这个值通常用来给出问题复杂性的上界和下界。

注意：$X_n^I$表示在大小为$n$的结构中插入一个元素或在第$n+1$个位置插入一个元素所需要的操作次数。

虽然这些度量是随机变量（因为它们依赖于度量它们时的特殊结构），但$C_n$和$C_n'$是例外，在大多数文献中认为它们表示的是期望值。

## 1.3 概率

一个给定事件的概率记为$Pr\{事件\}$。随机变量按前一节所叙述的来规定。随机变量$X$的期望值记为$E[X]$，其方差记为$\sigma^2(X)$。

尤其是对于离散变量$X$

$$E[X]=\sum_i iPr\{X=i\}$$

$$\sigma^2(X)=\sum_i i^2Pr\{X=i\}-E[X]^2=E[X^2]-E[X]^2$$

我们将明确指出计算期望值时所涉及到的概率变域。这个问题在某些场合是有歧义的，并且是与期望值不可分割的一个问题。

为阐明这个问题而不使读者感到困惑，假设我们按键码填写一张散列表，然后想知道检索某一键码时所需的平均存取次数。我们有两个可能的概率变域：一个指选择作为检索的键码（填表时第一个插入的键码，第二个插入的键码，等等），另一个指这些键码的实际值，或者是它们的探测序列。我们可以计算相对第一个变域、第二个变域或两者兼而有之的期望值。简言之，就是我们可以求出某一给定文件里任一键码的期望值，或求出任一文件里某一给定键码的期望值，或求出任一文件里任一键码的期望值。

除非另外声明，(1) 元素均为随机独立均匀分布$U(0,1)$；(2) 在所有可能的元素中，对某一给定元素的选择是均匀离散的；(3) 与多重域有关的期望值是相对这个多重域来计算的。按照上面的例子，就是要计算从均匀$U(0,1)$分布中任意选取的变量的期望值。

## 1.4 渐近符号

这本手册中大多数的复杂性度量按照问题的规模来讲都是渐近的。我们所使用的渐近符号是相当标准的，现给出如下：

$$f(n)=O(g(n))$$

表示：存在一个$k$和一个$n_0$，使得对$n>n_0$，有$|f(n)|<kg(n)$。

$$f(n)=o(g(n))\rightarrow\lim_{n\to\infty}\frac{f(n)}{g(n)}=0$$

$$f(n)=\theta(g(n))$$

蕴含：存在$k_1$，$k_2$（$k_1\times k_2>0$）及$n_0$，使得对$n>n_0$，有$k_1g(n)<f(n)<k_2g(n)$；等价地，即$f(n)=O(g(n))$及$g(n)=O(f(n))$。

$$f(n)=\Omega(g(n))\rightarrow g(n)=O(f(n))$$
$$f(n)=\omega(g(n))\rightarrow g(n)=O(f(n))$$
$$f(n)\approx g(n)\rightarrow f(n)-g(n)=O(g(n))$$

我们将把算术运算与阶的符号一起自由使用，例如

$$f(n)=h(n)+O(g(n))$$

意味着

$$f(n)-h(n)=O(g(n))$$

当我们写出$f(n)=O(g(n))$时，都意味着我们不知道更好的渐近的界，即不知道存在着$h(n)=O(g(n))$使得$f(n)=O(h(n))$。

## 1.5 关于程序设计语言

我们使用Pascal和C两种语言来对算法进行编码。在写出许多算法的代码之后，我们还是发现在有些情况下这两种语言都不能给出非常精巧的或可读性好的代码。因此，在任何可能的场合，我们将使用能给出最短的和可读性最好的代码的语言。我们有意让Pascal编码风格与C编码风格相互模仿。

少数Pascal程序含有goto语句。这些语句用来替换相等价的C语句return和break，并且相应地加以注解。我们确实认为Pascal中没有return和break是这个语言的一个缺陷。在用Pascal对某些算法进行编码时另一个令人烦恼之处，是它计算逻辑表达式时没有顺序。但是有这样的特征将使算法容易读懂。典型的“绊脚石”是

**while**($p\neq nil$)**and**($key\neq p\uparrow.key$)**do**$\cdots$

在C中如果使用这个序列和运算符(&&)，这样的语句是行得通的；但在Pascal中，我们则必须使用

**while** $p\neq nil$ **do begin**

```
if key = p↑.k then goto 999{* * * break * * *};
...
999:
```

其他一些微小的缺陷是：不能计算Pascal中非堆的目标的地址（这使得键表的处理更加困难）；C中缺少with语句和var参数(尽管这在技术上是可以克服的，但使算法难以理解)。

我们所用的Pascal代码尽可能与K. Jensen和N. Wirth所著«Pascal User Manual and Report»中所描述的语言保持一致。C编码则与B.W.Kernighan和D. M.Ritchie所著«The C Programming Language»中所描述的语言保持一致。

## 1.6 关于算法的代码

除去极少数明显地用伪代码写成的算法外，这本手册中的算法都在两种不同的编译程序下运行和测试过。事实上，印出来的正文程序就是用来编译、测试、模拟运行和计时的程序。这样做的目的是为了消除（或至少显著减少!）错误。

每一类算法都有一个"测试程序集"，它不仅用来检查算法的功能是否正确，还用来检查极限条件的处理是否合适（例如，排序程序是否能对空文件、只有一个元素的文件、以及所有键码均相等的文件排序？等等)。

大多数情况下，算法以一个函数、一个过程、或一组函数或一组过程的形式给出。有些情况下，对于非常简单的算法，代码可写成直接插入的形式；这些代码可以封装在一个过程中，或插入到另一段代码中。

有些算法，特别是搜索算法，是构成其他算法或程序的程序块成分。一些标准动作不应由算法本身来说明，但一旦这个算法要与其他部分"合成"时则应对它进行说明（第2章要详细定义合成的概

念)。标准动作的一个典型例子是出错条件。这本手册中的算法代码对于这些标准动作总是使用相同的名字。

执行中出现非期望的情况时的出错检测。每当出错时，可由任意一段语句来取代。例如测试程序打印出一条适当的信息。

完成一次成功搜索所执行的 found(record) 函数调用。其变量为包含所给键码的记录或指向该记录的指针。

不成功搜索调用的 notfound(key)函数。其变量为未找到的那个键码。

我们已经进行了很大的努力以避免对相同条件下的这些标准动作进行重复编码。这使得更易于用代码块来代替这些标准动作。

## 1.7 复杂性度量和实际计时

对于某些算法族，我们包括了实际计时比较。理解这些计时值时要注意，因为它们仅仅反映了硬件、编译程序、操作系统等很多因素中的一个采样点。然而我们有同样有力的理由来给出至少一组实际复杂性的度量值。

列出实际计时比较的主要原因是：

(1) 它们体现了操作的实际花费；

(2) 它们体现了一些隐含的花费，如存储分配、索引等。

主要的缺点、或可能使这些实际计时失效的因素是：

(1) 测试结果依赖编译程序：虽然对于每一种语言用的是同一个编译程序，但一个编译程序可能对某些构造比对另一些构造有利；

(2) 测试结果依赖硬件；

(3) 某些情况下，当要大量使用存储空间时，计时或许要依赖装载。

计时是在运行 Berkeley Unix 4.1 操作系统的 VAX 11/780 上

进行的。使用C和Pascal编译程序的同时，还使用了优化程序或目标代码改进程序，以获得算法的最好实现。

我们没有试图对不同语言编的程序来比较计时。所有的计时结果都是根据最快的算法计算出来的。我们选择在较大规模的问题上进行测试，这样可避免启动花费和装载等因素的影响。在这些情况下，一些 $O(n^2)$ 算法的运行特性显得很糟糕。