排序和查找理论及算法

周建钦
赵志远 编著

科学出版社

排序和查找理论及算法

周建钦　赵志远　编著

科学出版社

1993

（京）新登字 092 号

内 容 简 介

本书主要讲述排序和查找理论及算法研究，书中详细讨论了应用概率统计理论的排序和查找算法.

本书首先介绍以比较为基础的传统排序和查找理论及四个简单实用算法，然后重点阐述以概率分布为基础的排序和查找理论，并介绍了平均工作量均为 $C(N)$ 的七个快速排序算法和八个快速查找算法，每个算法均给出其设计思想、算法描述、算法分析和算法的应用子程序.

本书力求深入浅出，理论与算法并重，为便于读者学习，书中还提供了C语言子程序.

本书可作为高等院校计算机专业师生、从事计算机研究工作人员的参考书，也可作为计算机软件设计人员的工具书.

排序和查找理论及算法

周建钦　赵志远　编著

责任编辑　鞠丽娜

科学出版社出版

北京东黄城根北街 16 号

邮政编码：100707

中国科学院印刷厂印刷

新华书店北京发行所发行　各地新华书店经售

*

1993 年 6 月第 一 版　开本：787×1092　1/32

1993 年 6 月第一次印刷　印张：4 1/8

印数：1—3 420　字数：90 000

ISBN 7-03-003417-1/TP·251

定价：4.20 元

前　言

排序和查找在计算机系统软件和应用软件设计中使用频率非常之高，因而对排序和查找算法的研究经久不衰．计算机科学家 D. E. Knuth 在他的专著《排序和查找》中指出："我确信，程序设计的每一个重要方面，实际上都离不开排序或查找"；"计算机运行时间的 1/4 以上是花在排序上，有许多计算机装置，排序竟用去计算机时间的一半以上."

早在计算机问世之前，排序和查找已有广泛应用，如士兵的队列，图书和档案的管理，还有最常用的字典．我们来看一下人的大脑是如何处理排序和查找的．士兵在组成队列之前，首先全体士兵根据自身的高矮估计各自应处的位置，组成队列时，起初在士兵之间都适当留有一点空隙，以方便其他士兵的插入．待全部士兵按高矮次序排定后，再紧凑一下即组成一个完美的队列．同样，查一个英语单词，人们并不是从字典的第一页查起，而是首先估计该单词的可能位置，然后再查找．若第一次查找失败，则再估计它的相对位置，如此反复，直到查找成功．这样的查找速度非常之快．但是，如果一个人用计算机中的二分查找法查字典，速度一定慢得惊人．这里很重要的一点是，人的大脑处理问题时具有整体思想，即把所有问题对象看作一个相互联系的整体．我们相信，将这种整体思想应用于软件设计，能够减少软件的复杂性或改善软件的其他性能．

本书试图将这种整体思想应用于排序和查找．我们认为

计算机中处理的数据具有概率分布规律，因而研究设计处理这些数据的算法可以利用数据的概率分布规律；对于常用的算法，我们认为平均复杂性是衡量算法好坏的主要尺度。

传统的以比较为基础的排序和查找算法，其平均工作量最少为 $O(N\log N)$. 利用数据的概率分布信息设计排序和查找算法，其平均工作量可减少到 $O(N)$. 这方面的研究得到国内外计算机工作者的普遍关注，并已取得许多研究成果。

本书分为两大部分。第一部分阐述以比较为基础的传统的排序和查找理论，叙述四个简单实用的传统算法：简单插入排序，快速排序，顺序查找和二分查找。第二部分阐述以概率分布为基础的排序和查找理论，叙述了七个快速排序算法和八个快速查找算法，它们的平均工作量均为 $O(N)$. 对书中的每个算法都先阐述其设计思想，然后进行算法描述、算法分析，最后给出算法的应用子程序。

由于计算机的高速发展，内存容量日益增大，以至在多数情况下，可以在计算机的内存中实现排序和查找，因而本书讨论的排序和查找只限于内部排序和查找。如果需要，读者可直接将本书中的排序和查找算法应用于外部排序和查找的相应部分。基于同样的理由，本书着重考虑算法的时间复杂性，即算法的工作量，较少讨论算法的空间复杂性。

本书在内容上力求做到深入浅出，理论与算法并重，并提供C语言子程序供用户直接使用。本书可作为高等院校计算机专业师生、计算机研究人员的参考书，也可作为软件设计者的工具书。

复旦大学李贤平教授和山东矿业学院吴哲辉教授详细审阅了初稿，并提出了许多宝贵意见。复旦大学刘克俭硕士对部分数学证明提出了宝贵意见。在程序调试过程中，山东潍

坊计算机公司姚刚同志做了很多工作．编写过程中得到山东矿业学院李成尧副教授，崔先国讲师和张明金讲师等的支持和帮助，谨在此一并表示感谢。

作者

1991年7月

目　录

• ▼ •

第一章　排序和查找的工作量

1.1　排序和查找

排序和查找是计算机科学中非常基本且使用频繁的运算，在计算机系统软件和应用软件中都有广泛的应用。排序和查找是相互独立而又密切联系的两个问题。

1.1.1 排序

排序（sorting）又称分类，是将一组数据按规定的顺序重新排列存放，其定义如下：

设给定待排序的N个记录为

$$R_1, R_2, \cdots, R_N$$

我们称这N个记录的整个集合为一个文件。每个记录 R_i 有一个键 K_i，称为关键字，它支配着排序过程。

诸关键字之间存在一个次序关系"$<$"，使得对于任意三个关键字 a，b，c，下列条件成立：

i) $a<b, a=b, b<a$ 三个可能性中恰有一个可能性成立（三分律）；

ii) 如果 $a<b$，$b<c$，则 $a<c$（传递律）。

这两个性质表示了次序关系，即全序。

所谓排序就是确定N个记录的一个排列 $R_{p(1)}, R_{p(2)}, \cdots, R_{p(N)}$，其中 $p(1), p(2), \cdots, p(N)$ 是 $1, 2, \cdots, N$ 的一个排

列，它以非递减的次序存放诸关键字：

$$K_{p(1)} \leqslant K_{p(2)} \leqslant \cdots \leqslant K_{p(N)}$$

1.1.2 查找

查找（searching）又称为搜索，其定义如下：

假设有N个记录组成的文件 F，每个记录 R_i 有一个键 K_i，查找一个记录 R，即是在文件F中寻找以K为键的那个记录，其中K是R的键。显然查找完成之后有两种可能性：已找到以K为键的一个记录，称查找成功；未找到以K为键的记录，称查找失败。

如果一种查找方法要求被查找的文件是有序的，则这种查找方法称为有序查找；如果不要求被查找文件有序，则称为无序查找。

如果在排序（查找）过程中只使用关键字的比较和复写运算，则称这类排序（查找）方法为比较排序（查找）。这类方法只利用了关键字的序的信息。如果排序（查找）时不仅利用关键字的序的信息，而且还利用关键字的值的信息（概率分布规律），则称这类排序（查找）方法为分布排序（查找）。

1.2 比较排序和比较查找的工作量下界

1.2.1 算法复杂性

对排序和查找算法，我们最关心的是它们的工作量。算法的工作量，也称为算法的时间复杂性，一般有三方面的指标：

i）最坏情况下的工作量；

ii）平均情况下的工作量；

iii）最好情况下的工作量。

由于排序和查找算法使用频繁，因此衡量算法好坏的主要指标应是算法在平均情况下的工作量，即平均复杂性。快速排序（quick sort）被公认为是一个快速算法，就是因为它的平均工作量较少，为 $O(N\log N)$。事实上，快速排序的最坏情况下的工作量很大，为 $O(N^2)$。

1.2.2 比较排序算法的工作量下界

对于比较排序和比较查找算法，其算法的工作量主要决定于比较和交换的次数。

设 N 个互不相等的关键字为 $X_1, X_2, \cdots, X_N$，我们将每一个排序算法与一个二叉判定树对应起来。设 A 是一个排序算法，对于给定的文件大小 N，二叉判定树的定义如下：

（1）若算法 A 第一次将 X_{i_0} 和 X_{j_0} 相比较，则二叉树根的标记为 (i_0, j_0)。

（2）假设某一顶点的标记为 (i,j)，则顶点 (i,j) 的左儿子的标记是当 $X_i < X_j$ 时，算法接下去要比较的两个键的下标；顶点 (i,j) 的右儿子的标记是当 $X_i > X_j$ 时，算法接下去要比较的两个键的下标。

（3）若在比较 X_i 和 X_j 之后算法停止，顶点 (i,j) 的左儿子对应当 $X_i < X_j$ 时算法输出的一个关键字排列；顶点 (i,j) 的右儿子对应当 $X_i > X_j$ 时算法输出的一个关键字排列。这时顶点 (i,j) 的左、右儿子的度数都为 1，因而称为判定树的叶子。

如果根到顶点 V 的路的长度为 d，则称顶点 V 在判定树的第 d 层。若判定树共有 h 层，则称判定树的高度为 h。

排序一个输入文件，对应于判定树中从根到一片叶子的一条路，而比较次数即是这条路的长度。因而算法在最坏情

况下的比较次数即是对应判定树的高度；算法的平均比较次数，即是对应判定树中所有从根到叶子路的平均长度．N个互不相等的关键字有 $N!$ 种排列，因而每一棵判定树都恰好有 $N!$片叶子，即从根到叶子有 $N!$条路．

引理 1.1 设 l 是一棵二叉树的叶子数，h 是它的高度，则 $l \leqslant 2^h$．

证明 对 h 直接用归纳法可证．

引理 1.2 对于给定的 N，任何一个比较排序算法的判定树的高度至少为 $\lceil \log_2 N! \rceil$，其中 $\lceil X \rceil$ 表示取大于 X 的最小整数．

证明 判定树的叶子数为 $N!$因为

$$N! \leqslant 2^h$$

即

$$h \geqslant \log_2 N!$$

又因 h 是一个整数，所以

$$h \geqslant \lceil \log_2 N! \rceil$$ □

于是，在最坏情况下，排序所需的比较次数至少为

$$\lceil \log_2 N! \rceil$$

由 Stirling 公式：

$$N! \doteq \sqrt{2\pi N}\left(\frac{N}{e}\right)^N$$

得到

$$\log_2 N! = N\log_2 N - N\log_2 e + \frac{1}{2}\log_2 N + O(1)$$

显然

$$h \geqslant O(N\log_2 N)$$

定理 1.1 通过比较排序 N 个元素，最坏情况下比较次数至少为 $O(N\log_2 N)$．

排序作为一种常用的运算，我们更关心它在各种情况下比较次数的平均，即平均比较次数．下面讨论比较排序的平均比较次数．

我们知道在一棵判定树中，每个顶点的度数是 0 或者是 2，具有这个性质的二叉树称为 B 树． 用 epl(external path length) 表示一棵树中所有从根到叶子路的长度之和．

引理 1.3 在具有 l 片叶子的 B 树中，如果所有的叶子在至多 2 个相邻的层上，则 epl 取最小值．

证明 如图 1.1，若有一棵深度为 d 的 B 树 T，它有一片叶子 X 在第 k 层，$k \leqslant d-2$，我们将给出具有同样叶子数的 B 树 T'，但它具有较小的 epl． 在第 $d-1$ 层选择一个不是叶子的顶点 Y，将它的两个儿子移去，而加在顶点 X 上．这时总的叶子数不变．我们不再计算到 Y 的儿子的路和到 X 的路使 epl 减少 $2d+k$，但计算到 Y 的路和到 X 的儿子的路使 epl

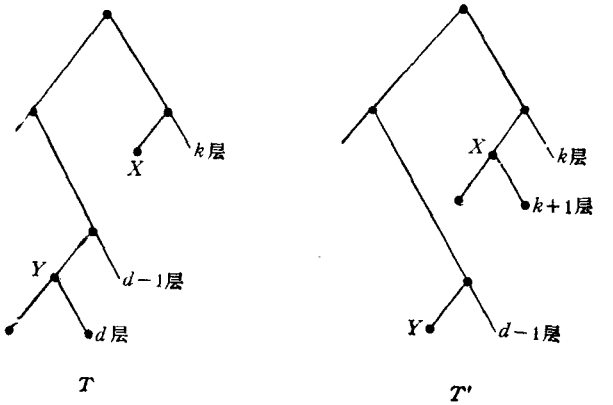

图 1.1

增加 $2(k+1)+d-1=2k+d+1$． epl 净减少量为

$$2d + k - (2k + d + 1) = d - k - 1 > 0 \qquad \square$$

引理 1.4 令 $\lfloor X \rfloor$ 表示取小于 X 的最大整数，则具有 l 片叶子的 B 树的最小 epl 是：

$$l\lfloor \log_2 l \rfloor + 2(l - 2^{\lfloor \log_2 l \rfloor})$$

证明 由引理 1.3，我们不妨设 B 树的所有叶子至多在两层上。

i) 若 $l = 2^k$，则 $k = \log_2 l$，所以

$$\text{epl} = l \cdot k = l\lfloor \log_2 l \rfloor + 2(l - 2^{\lfloor \log_2 l \rfloor})$$

ii) 若 l 不是 2 的幂，则树的高度 $h = \lceil \log_2 l \rceil$。从根到所有叶子的路至第 $h-1$ 层长度之和为 $l(h-1)$，为了得到 epl，对于第 h 层的每一片叶子，必须加上 1。在第 $h-1$ 层每一个不是叶子的顶点，在第 h 层有两片叶子。设第 $h-1$ 层的叶子数为 X，第 h 层的叶子数为 Y，则

$$X + Y = l$$

$$X + \frac{1}{2} Y = 2^{h-1}$$

所以

$$Y = 2(l - 2^{h-1})$$

并得到

$$\begin{aligned} \text{epl} &= l(h-1) + 2(l - 2^{h-1}) \\ &= l\lfloor \log_2 l \rfloor + 2(l - 2^{\lfloor \log_2 l \rfloor}) \end{aligned} \qquad \square$$

定理 1.2 通过比较排序 N 个元素，各种情况下平均比较次数至少为 $O(N \log N)$。

证明 最小平均比较次数是对应 B 树的最小 epl/$N!$，即

$$\frac{N!\lfloor \log_2 N! \rfloor + 2(N! - 2^{\lfloor \log_2 N! \rfloor})}{N!} = \lfloor \log_2 N! \rfloor + \varepsilon$$

因为 $N! - 2^{\lfloor \log_2 N! \rfloor} < N!/2$，故 $\varepsilon < 1$。再使用 Stirling 公

式，即得欲证结论。 □

定理 1.1 告诉我们，比较排序的最坏情况下的工作量下界为 $O(N\log N)$。定理 1.2 告诉我们，比较排序的平均工作量下界也是 $O(N\log N)$。

1.2.3 比较查找的工作量下界

设 N 个互不相等的关键字为 $X_1, X_2, \cdots, X_N$，我们将每一个查找算法与一棵二叉树对应起来。设 A 是一个查找算法，对于给定的文件大小 N，二叉判定树的顶点用 1 到 N 之间的数标记，规则如下，设待查找记录的键为 X：

（1）树根的标记是算法 A 第一次与 X 比较的键的下标。

（2）假设某一顶点的标记是 i，则顶点 i 的左儿子的标记是当 $X < X_i$ 时，算法接下去要与 X 比较的键的下标；顶点 i 的右儿子的标记是当 $X > X_i$ 时，算法接下去要与 X 比较的键的下标。

（3）若在比较 X 和 X_i 之后，算法停止，则顶点 i 没有分支。（这时若 $X > X_i$ 或 $X < X_i$，则查找失败；若 $X = X_i$，则查找成功。）

引理 1.5 高度为 h 的二叉判定树至多有 $2^{h+1}-1$ 个顶点。

证明 若高度为 h 的二叉判定树 T 的顶点最多，则 T 必定是完全二叉树，这时 T 的顶点数为

$$1+2+2^2+\cdots+2^h=2^{h+1}-1$$ □

给定一个输入，算法 A 将沿着其判定树中从根出发的某一条路，执行所指明的比较。比较次数即是这条路上的顶点个数，算法 A 在最坏情况下的比较次数是从根到叶的最长路上的顶点个数，即树的高度＋1。

我们断定二叉判定树至少有N个顶点．用反证法证明．假设树中的顶点个数小于 N，则对于1到N之间的某个 i，树中没有一个顶点的标记为 i．这时，我们可以构造两个输入文件F和 F'，使得对于$1\leqslant j\leqslant N$且 $j\neq i, X_j = X'_j \neq X$，但 $X_i = X$ 且 $X'_i \neq X$．因为在判定树中没有一个顶点的标记是 i，所以算法A从来没有将X和 X'_i 相比较．由于两个输入中的其余记录相同，故对两个输入，算法A的执行是一样的，且给出同样的输出．因此至少对于一个输入文件，A的输出是错误的，即算法A是错误的．由此可以得出结论，判定树中至少有N个顶点．

由引理1.5可知，$N \leqslant 2^{h+1} - 1$，其中 h 是树的高度，所以

$$h \geqslant \lceil \log_2(N+1) \rceil - 1$$

又因

$$\lceil \log_2(N+1) \rceil = \lfloor \log_2 N \rfloor + 1$$

所以

算法A在最坏情况下的比较次数至少是 $\lfloor \log_2 N \rfloor + 1$．

由算法A的一般性，可知下述定理．

定理1.3 通过比较在大小为N的文件中查找一个记录，最坏情况下比较次数至少为 $O(\log N)$．

1.2.4 比较排序和比较查找的工作量

这一节我们统一讨论排序和查找的工作量，即比较次数．为此首先定义查找算法的工作量为：在大小为N的文件中分别查找文件中的N个记录所需的工作量．（以下的查找算法工作量或比较次数含义与此相同．）

设文件F中N个互不相等的关键字依次为 $X_1, X_2, \cdots, X_N$，对文件F排序后，N个关键字依次为 $X_{p(1)}, X_{p(2)}, \cdots,$

$X_{p(N)}$，其中 $p(1),p(2),\cdots,p(N)$是 $1,2,\cdots N$，的一个排列，因而排序共有 $N!$ 种可能结果。

设文件 F 中N个关键字互不相等，重新以递增次序排列为： $X_1, X_2, \cdots, X_N$。设 X_i 在 F 中的位置为 $p(i)$，则 $p(1),p(2),\cdots,p(N)$ 是$1,2,\cdots,N$的一个排列，因而查找共有 $N!$ 种可能结果。

一次比较即是作出一个“是或非”的判断，因而比较排序和查找可归结为一个较一般的问题： 设有一个含有 M 个元素的集合 S， A 任取 S 中的一个元素，B 向 A 提出至多 K 个属于“是或非”的问题，判断出 A 取的是什么元素，问 K 的下界是多少？

第一次是非问题把 S 分成两个子集合，一个子集合对应“是”的元素，另一个子集合对应“非”的元素。根据抽屉原理，其中至少有一个子集合 $A_1^{(1)}$：

$$|A_1^{(1)}| \geqslant \frac{M}{2}$$

另一个子集合 $A_2^{(1)}$：

$$|A_2^{(1)}| \leqslant \frac{M}{2}$$

设正确的答案属于 $A_1^{(1)}$，B 提出的第 2 个问题又把 $A_1^{(1)}$ 分成两部分，其中之一 $A_1^{(2)}$ 有

$$|A_1^{(2)}| \geqslant \frac{M}{4}$$

为了得出正确的答案，B 必须设法把范围缩小到一个元素。所以

$$\frac{M}{2^K} \leqslant 1, \text{ 即 } K \geqslant \log_2 M$$