一种线性原地二路归并算法

范时平　汪林林　何先刚
（重庆邮电学院　重庆400065）

摘　要　和其它排序算法相比，二路归并最适合于两个有序子表的排序。但经典原地二路归并算法的时间性能是乘积型的，尚有改进空间。文章介绍了改进经典原地二路归并算法所需的基本技术，提出了一种线性原地二路归并算法。归并长度分别为 m 和 n 的两个有序子表，该算法最多需要 $2.5m+1.5n+4.5\sqrt{m+n}$ 次比较和 $8m+7n-3\sqrt{m+n}$ 次移动。

关键词　原地算法，二路归并，块交换，内部缓冲，分块，块排序

A Linear-Time In-place Merging Algorithm

FAN Shi-Ping　WANG Lin-Lin　HE Xian-Gang
(Chongqing University of Posts and Telecommunication, Chongqing 400065)

Abstract　Comparing with other sorting algorithms, the 2-way merge algorithm is the best one to sort two sorted sublists. But the time complexity of the lassic 2-way merge algorithm is the product of the two sorted sublists that leaves something to be desired. The basic techniques needed for improving the classic 2-way merge algorithm are introduced, and a linear-time in-place 2-way merge algorithm is developed. The algorithm needs $2.5m+1.5n+4.5\sqrt{m+n}$ comparisons and $8m+7n-3\sqrt{m+n}$ assignments to merge two sorted sublists of lengths m and n.

Keywords　In-place algorithm, 2-way merge, Block swapping, Internal buffering, Partting data, Block sorting

1　引言

给定一个长度为 n 的顺序表，每个元素由一个关键字和其它的相关信息构成，排序算法的任务就是根据关键字以非降序（或非升序）重新安排数组中的元素（不失一般性，以下按非降序排序）。排序算法仅允许做关键字比较和元素移动两个操作，并用关键字比较次数和元素移动次数衡量排序算法的时间性能和空间性能。

如果一个算法所使用的额外空间是一个固定常数，与处理问题的规模无关，则我们称该算法是原地算法。

归并排序是基于分治策略的一种算法，其基本思想是把若干有序子表合并为一个有序表。二路归并则是将两个有序子表合并成一个有序表。

二路归并最适合于两个有序子表的排序[1]，将长度分别为 m 和 n 的两个有序子表归并为一个有序表，经典原地二路归并算法需要 $O(m+n)$ 次元素比较和 $O(m\times n)$ 次元素移动[2]，其时间性是乘积型的，还有很大的改进空间。本文介绍了改进经典原地二路归并算法所需的基本技术，并提出了一种线性原地二路归并算法。

为便于叙述，我们把线性原地二路归并算法的设计分为3个阶段：准备（包括确定数据块大小、构造内部缓冲区等）、非缓冲区元素排序、内部缓冲区元素排序。本文第2节介绍新算法使用的基本技术，第3节介绍新算法，第4节验证算法正确性并分析算法性能，最后为总结。

2　基本技术

2.1　数据块交换

根据文[3]，交换长度分别为 j 和 k 的两个相邻数据块，移动元素次数为 $j+k+\gcd(j,k)\leqslant j+k+\min(j,k)$（这里 $\gcd(j,k)$ 表示 j 和 k 的最大公约数）。交换两个长度为 L 的等长数据块，需要 $3L$ 次元素移动。

2.2　内部缓冲

用两个子表自身元素占据的存储空间作为排序的工作空间，并称其为内部缓冲区。在使用内部缓冲时，要遵守以下几个原则：

1）内部缓冲区的元素由两个有序子表中关键字最大（或最小）的若干个元素构成。

2）内部缓冲区的元素构成后，应将其移到有序子表的某一端，否则会增加不必要的元素移动。

3）将内部缓冲区元素分插到未归并区域时，不影响未归并元素的相对位置。

在归并元素时，内部缓冲区的元素会分插到有序子表的未归并区域，但绝对不能影响未归并元素的相对位置。否则将破坏未归并块内部元素的有序性，也就破坏了二路归并要求两个待排序子表都有序的基本条件，无法保证以后归并每个元素都在常数时间内完成。

4）内部缓冲区的尺寸选择要适当。

内部缓冲区不能太大，必须将缓冲区的构建和缓冲区元素排序的总时间开销控制在线性阶。内部缓冲区也不能太小，要大到满足上述原则三为前提。一般情况下非缓冲排序的内部缓冲区块与数据块的尺寸相等（例外见文[4,5]，但其思想与本文迥异）。

2.3　划分数据与块排序

范时平　讲师，主要研究方向为算法设计与分析、面向对象技术、计算机网络等。汪林林　教授，硕士研究生导师，主要研究方向为数据库系统、GIS系统、数据仓库、算法设计与分析、计算机网络等。

将两个有序子表各分成若干块，并将关键字相近的块调整到一起，缩小元素比较和移动的范围，以减少元素比较和移动的次数。把一个块内元素的最大关键字作为整个块的关键字，对数据块进行排序的过程称为块排序。数据块的尺寸根据待归并有序子表的长度和所采取的归并技术确定。普通数据块长度相同，并且与内部缓冲区的尺寸相匹配。如果两个待归并有序子表产生缓冲区后剩余元素个数不是所确定数据块长度的整数倍，就会各产生一个不规则块，其初始位置由归并技术确定。

3 线性原地二路归并算法

利用前面介绍的技术，我们设计了一个线性原地二路归并算法。

1)初始化和特殊情况处理

(1) 如果待排序表本身是有序的，归并结束。

(2) 计算子表1的长度 m 和子表2的长度 n。

(3) 确定缓冲区和数据块的长度 $Blen:\lceil (m+n)^{1/2}\rceil$。

(4) 如果 $n\leqslant Blen$，则用相邻块交换技术将子表2元素分插到子表1中，归并结束。

(5) 如果 m 和 n 恰好是 $Blen$ 的整数倍，则两个子表的首块长为 $Blen$；否则首块为不规则块，长度分别为 $m\%Blen$ 和 $n\%Blen$(%为模运算符)。我们还要记住两个首块的位置，以便对不规则首块作特别处理。

2)构造内部缓冲区

(1)选取子表1和子表2中最大的 $Blen$ 个元素构成内部缓冲区。

(2)将内部缓冲区移到最左端，保持其它元素的相对位置。

3)划分数据和块排序　根据块长度 $Blen$，从前向后扫描并划分两个子表。划分的同时对数据块按非递减序排序(块内元素最大关键字相同时，以块内元素最小关键字作为辅助关键字)。步骤如下：

(1) 如果两个子表的首块是不规则块，用相邻数据块交换技术将两个子表的首块先排好，同时修改被移动首块的起始位置。
(2) 找出子表1未排序的最小块，将其与子表2的当前块比较。
① 如果子表2的当前块小，则用等长块交换技术将其交换到前面，这会破坏子表1中未排序块的有序性；否则转②。
② 如果子表1的当前块是子表1未排序的最小块，不用交换；否则用等长块交换技术将子表1的当前块与子表1未排序的最小块交换。
(3) 重复(2)直到子表1或子表2排完。
(4) 如果子表1未排完，则找出子表1未排序的最小块。如果子表1的当前块是子表1未排序的最小块，不用交换；否则用等长块交换技术交换子表1的当前块与子表1未排序的最小块。
(5)重复(4)直到子表1排完。

4)非缓冲区数据排序

(1)确定当前子序列1和当前子序列2。
当前子序列1开始于最左边的未归并数据块，终止于数据块 i($1\leqslant i$)，数据块 i 是满足其块尾元素关键字大于数据块 $i+1$ 的块首元素关键字的第1个块。当前子序列2只有数据块 $i+1$。
(2)利用内部缓冲区将子序列2的部分元素分插到子序列1中，步骤如下：
① 如果子序列1的当前最左元素小，将其与缓冲区当前最左元素交换；否则将子序列2的当前最左元素与缓冲区当前最左元素交换。
② 重复①直到子序列1的元素处理完。
(3)将(2)处理后子序列2的剩余部分作为1个数据块。重复(1)、(2)直到不能找到当前子序列1和当前子序列2。
(4)将前3步处理后剩余的非缓冲区元素与缓冲区元素交换。此时非缓冲区元素已经排好序，无序的缓冲区元素位于表右端。

5)缓冲区数据排序　用选择排序算法对非缓冲区元素排序，归并结束。

4 算法的正确性验证与性能分析

通过比较相邻元素的关键字，证明我们的线性原地二路归并算法是正确的。

算法中的每一步都可避免使用递归；用指针向函数传递待归并表首地址只需要几个字节，其它参数就是几个下标值；每个函数内部都只需要有限的空间用于存储下标和辅助实现数据块交换，因此算法的空间性能是原地的。

当待归并表是正序时，算法的时间性能最好，比较1次，移动0次。下面我们重点分析最坏情况下算法的时间性能。

1)$n\leqslant Blen(Blen<\sqrt{m+n}+1)$

如果每次只归并子表2中的1个元素并且子表1中的所有元素都被后移时，移动次数最多。此时子表1中的元素最多共后移 m 次。将子表2的倒数第 i 个元素移到最终位置最坏情况下要将子表2中的前$(n-i)$个元素和子表1未归并的后 $k(1\leqslant k\leqslant m)$个元素交换，辅助移动次数为 $\gcd(k,n-i)\leqslant(n-i)$，即子表2中元素最多移动次数为$2\sum_{i=1}^{n}(n-i)=n(n-1)\leqslant m+2\sqrt{m+n}$。综上，元素移动总次数不超过$2m+2\sqrt{m+n}$。由于每次比较至少归并1个元素，所以最多比较 $m+n-1$次。

2)$n>Blen(\sqrt{m+n}\leqslant Blen<\sqrt{m+n}+1)$

构造内部缓冲区后子表1有 $B1\leqslant\lceil\frac{m}{Blen}\rceil<\frac{m}{\sqrt{m+n}}+1$块，子表2有 $B2\leqslant\lceil\frac{n}{Blen}\rceil<\frac{n}{\sqrt{m+n}}+1$块，显然 $B1+B2\leqslant\sqrt{m+n}$。在构造块内部缓冲区阶段，最多比较 $Blen<\sqrt{m+n}+1$次。在块排序阶段，当子表1还有 i 个块未归并时，在子表1中查找最小块最多比较$2(i-1)$次，即在子表1中查找最小块最多共比较$2\sum_{i=1}^{B1-1}(i-1)=(B1-1)(B1-2)<m$ 次；在两个子表的最小块中确定当前最小块最多共比较$2(B1+B2-1)<2\sqrt{m+n}$次。所以块排序阶段最多比较 $m+2\sqrt{m+n}$次。非缓冲区数据排序阶段，为确定当前子序列1和子序列2，除表的首块和尾块只比较1次外，其余块都比较2次时，所需的比较次数最多，为$2+2(B1+B2-2)<2\sqrt{m+n}-2$。非缓冲区数据排序阶段，每次比较至少归并1个元素，最多比较 $m+n-\sqrt{m+n}$次。缓冲区元素排序比较$0.5(Blen^2-Blen)\leqslant 0.5(m+n+\sqrt{m+n})$次。综上，当 $n>Blen$ 时，最多比较元素$2.5m+1.5n+4.5\sqrt{m+n}$次。

在构造缓冲区阶段，最多移动 $m+n+\sqrt{m+n}+1$次元素。设子表2的首块长度为 $i(1\leqslant i\leqslant\sqrt{m+n}+1)$，则在块排序阶段将子表2的首块移到子表1前面时移动最多移动元素 $m+2i-\sqrt{m+n}-1$次；其余块排序最多移动元素$3(m+n-1-\sqrt{m+n}-i)$次，所以块排序最多移动元素$4m+3n-4\sqrt{m+n}-2$次。非缓冲区数据排序阶段归并1个元素最多移动3次元素，所以最多共移动元素$3(m+n-\sqrt{m+n})$次。缓冲区数据排序阶段最多移动$3\sqrt{m+n}$次元素。综上，当 $n>Blen$ 时，最多移动元素$8m+7n-3\sqrt{m+n}$次。

下面我们通过实验分析算法的平均时间性能。由于 rand 函数产生的随机数在0到32767之间[6]，当子表很长时重复数据太多，随机性能不好。因此我们先用 rand 函数产生随机表，再用表中相邻元素的乘积替换表中元素。这样当表长在10^{10}范围内时，数据随机性很好。我们先将两个子表排好序，对相同的两个有序子表，用经典原地二路归并算法(以下简称经典

(下转第225页)

泡、折半插入、快速排序算法进行排序比较，各种排序算法所花费的时间如表1所示。

表1 Turbo PASCAL 四种排序方法的排序时间对比(单位:秒)

数据个数 N	冒泡排序	快速排序	折半插入	精度归"档"插入排序
1000	0.54	0.02	0.18	0.02
5000	2.30	0.06	0.56	0.03
10000	6.09	0.08	2.14	0.06

观察分析上述实验结果，可以看出:精度归"档"插入排序的时间复杂度应处在高效排序算法类中，且数据越多，效率越高。

结束语 本文提出的精度归"档"插入排序算法对于均匀分布或非均匀分布的数据最为有效，其时间复杂度仅为 $O(n)$。但是，对于极不均匀分布的数据(绝大多数数据落在同一"档"里)，该排序方法的效率将明显下降，这时排序的时间复杂度变为 $O(n^2)$。好在这种情况在排序初期就能够预料到，以便转移到其它排序方法，从而可以避免排序效率的明显下降。

参考文献

1 严蔚敏，吴伟民. 数据结构[M]. 北京:清华大学出版社，1996.12

2 王晓东. 计算机算法分析与设计[M]. 北京:电子工业出版社，2001.114～146

3 王向阳. 均匀分布数据的分"档"统计插入排序算法研究. 数值计算与计算机应用，2000(9)

4 王向阳. 基本有序数据的分段堆排序算法研究. 小型微型计算机系统，1999(7)

5 王新刚. 有限次分组排序. 青岛大学学报，1999(3)

(上接第222页)

算法)和线性原地二路归并算法(以下简称新算法)分别处理。两个算法的元素比较次数、移动次数分别如表1、表2所示:

表1 经典算法与新算法的元素比较次数

子表1长	子表2长	经典算法比较次数 CO	新算法比较次数 CN	CO:CN
5000	5000	9995	17016	1:1.70
50000	50000	99998	173451	1:1.73
100000	100000	199998	348045	1:1.74
150000	150000	299997	521718	1:1.74
200000	200000	399999	696216	1:1.74
300000	300000	599995	1.0464e+006	1:1.74
500000	500000	999998	1.74347e+006	1:1.74
1000000	1000000	2e+006	3.49014e+006	1:1.75

表2 经典算法与新算法的元素移动次数

子表1长	子表2长	经典算法移动次数 MO	新算法移动次数 MN	MO/MN
5000	5000	1.26092e+007	70304	179
50000	50000	1.25474e+009	718890	1745
100000	100000	5.02698e+009	1.43383e+006	3506
150000	150000	1.12437e+010	2.16053e+006	5204
200000	200000	2.00146e+010	2.88428e+006	6939
300000	300000	4.51905e+010	4.32474e+006	10449
500000	500000	1.25073e+011	7.21951e+006	17324
1000000	1000000	4.9932e+011	1.44752e+007	34495

实验表明，新算法的平均元素比较次数比经典算法增加不到1倍，但新算法的平均元素移动次数却大幅减少，待排序序列越长，减少的比例越大。

为进一步考察新算法增加比较次数和减少移动次数的综合效果以及辅助操作对算法的影响程度，我们在赛扬1.7GCPU、256M 内存，Win98操作系统、VC++6.0环境下运行两个算法，运行时间如表3所示。

根据表3，新算法能够大大减少运行时间，待排序表越长，减少的比列越大。表3中新算法运行时间的减少比例小于表2中新算法元素移动次数的减少比列，这是由于新算法所增加元素的比较次数和大量辅助操作，部分抵消了减少元素移动次数的改进效果。

表3 经典算法与新算法的运行时间

子表1长	子表2长	经典算法运行时间 TO	新算法运行时间 TN	TO/TN
50000	50000	11秒	50毫秒	220
100000	100000	44秒	90毫秒	489
150000	150000	98秒	140毫秒	700
200000	200000	176秒	180毫秒	978
250000	250000	278秒	240毫秒	1158
300000	300000	395秒	280毫秒	1410
350000	350000	542秒	310毫秒	1748
400000	400000	706秒	380毫秒	1858
450000	450000	902秒	420毫秒	2148
500000	500000	1070秒	440毫秒	2431
1000000	1000000	8222秒	960毫秒	8565

总结 通过采用内部缓冲区、划分数据、对数据块进行排序等方法，我们设计了一个线性原地二路归并新算法。归并长度分别为 m 和 n 的两个有序子表($m \geqslant n$)，新算法最多用 $2.5m+1.5n+4.5\sqrt{m+n}$ 次比较和 $8m+7n-3\sqrt{m+n}$ 次移动。新算法将经典原地归并算法的移动次数从 $O(m \times n)$ 降低到 $O(m+n)$，从而大大缩短了算法的运行时间，待排序序列越长，算法改进效果越显著。因此该线性原地二路归并算法具有较高的理论和实用价值。

参考文献

1 严蔚敏，吴伟民. 数据结构(第二版)[M]. 北京:清华大学出版社，1991

2 Kruse R L, Ryba A J. DATA STRUCTURES AND PRORAM DESIGN IN C++(Copyright 1999)[M]. 北京:高等教育出版社(影印)，2001

3 Huang B-C, Langston M A. Fast stable merging and sorting in constant extra space[J]. The Computer Joural , 1992,35: 643～650

4 Geffert V, Katajainen J, Pasanen T. Asymptotically efficient in-place merging[J]. Theoretical Computer Science , 237(1-2):159～181

5 Symvonis A. Optimal Stable Merging[J]. The Computer Journal, 1995,38 : 681～690

6 Microsoft Visual C++6.0 MSDN 帮助信息