精度归"档"插入排序算法研究

王治和　贾俊杰

(西北师范大学数学与信息科学学院　兰州730070)

摘　要　提出了一种在最大值和最小值之间的数据范围内,由待排序数据的落点百分比精确到第一位小数点后经转换所形成的固定"档"位的基础上,利用归"档"统计和直接插入排序所形成的新排序算法—精度归"档"插入排序算法。概算法在待排序数据非极不均匀的情况下,时间复杂度降为$O(n)$,具有重要的实际意义。

关键词　排序,精度,归档,扫描,时间复杂度

Study of the Insertion Sort Algorithm of Filing according the Precision

WANG Zhi-He　JIA Jun-Jie

(College of Mathematics and Information Science, Northwest Normal University ,Lanzhou 730070)

Abstract　This paper puts forward a kind of new sort algorithm-study of insertion sort algorithm of filing according the precision that uses Filing Statistics and Straight Insertion Sort. In the course of the algorithm, unsorted data are transformed its percentage of placement-point which is calculated to one decimal place into file place between minimum and maximum . The time complexity of this algorithm is $O(n)$ when the unsorted data is not utmost non-uniform distribution. Therefore, this algorithm possesses the important meaning.

Keywords　Sorting,Precision, Filing, Scan, Time complexity

1　引言

排序是计算机数据处理经常使用的一种重要运算,寻找速度快、附加存储空间开销小的高效排序方法也一直是计算机领域内人们感兴趣的课题。到目前为止,通用的各种排序方法所能够达到的最佳时间复杂度为$O(n\log_2 n)$,且多半都是基于比较的排序方法[1,2]。也有人抛开比较操作,提出了新的排序算法,即根据概率分布的特点,将待排序关键码设计数学公式,如公式分组索引、一步到位排序等,其期望时间复杂度为[3~5]。

在实际应用中,待排数据一般为均匀分布,如成绩,年龄,工资等等,据此本文介绍了另一种排序算法,它根据待排序数据分布的特点,在最小值和最大值之间的区间范围内,由待排序数据的落点而提出一种谓之精度归"档"插入排序。该算法在附加$2n+22$个存储空间的情况下,时间复杂度降为$O(n)$,具有一定的实际意义。

2　精度归"档"插入排序算法描述

该算法是将记录按待排序数据不减的次序排列。算法用到4个数组:*data*、*value*、*adr* 和 *result*。*data* 为记录数组,它的记录类型有两个域 *key* 和 *file*,*key* 域存放初始时待排序数据,*file* 域统计各数据所归"档"数;*value* 用于统计各"档"数据个数;*adr* 用于存储各"档"首地址;*result* 用于存放排序结果。其中 *data* 和 *result* 的数组大小为 n,*value* 和 *adr* 的数组大小为11,因为共有11 个"档"位("档"位按0到10来划分)。

该算法的工作原理是:首先在待排序数据当中找出最大值 *Max* 和最小值 *Min*,计算每个待排序数据落在的区间中的百分比,可知$0\leqslant D=\frac{data[i]\cdot key-Min}{Max-Min}\leqslant 1$,据此取 D 小数点后第一位为所"归"档数,故可分为0、1、2、…、9、10等11"档",然后将待排序数据依次归"档",此时"档"间已排好序。根据数字精确度的概念,一般情况下落入同一"档"的数据较少,因此在"档"内用直接插入法排序,或者用其它传统排序方法亦可,这对时间复杂度的影响是微乎其微的。假设排序之前,待排序数据已经被读入数组 *data*,且 *file* 域的值初始化为0,其算法具体描述用类 *PASCAL* 编写如下:

1 [初始化数组 *value* 和 *adr*]

循环 i 以1为步长,从0到10,执行

$value[i]\leftarrow 0$;

$adr[i]\leftarrow 0$

2 [扫描数组 *data*,求出最大值 *Max* 和最小值 *Min*]从略。

3 [求出归"档"系数 L]

$L\leftarrow Max-Min$;

4 [确定 *data* 数组的 *file* 域的值]

循环 i 以1为步长,从1到 n,执行

$h\leftarrow\lfloor(\frac{data[i]\cdot key-Min}{L}+0.05)\times 10\rfloor$;

$data[i].file\leftarrow h$;

$value[h]\leftarrow value[h]+1$

5 [确定数组 *adr* 的值]

循环 i 以1为步长,从1到10,执行

$adr[i]\leftarrow adr[i]+value[i-1]$

6 [对满足 $value[i]\geqslant 2$的第 i"档"内数据进行局部的直接插入排序,然后将数据依次插入 *result* 中,此时待排序数据以递增顺序存放在数组 *result* 中]从略。

7 [算法结束]

下面,通过一个简单的具体实例来进一步详细说明精度归"档"插入排序算法的工作原理,这里假设有10个待排序随机整数{38,19,65,13,97,49,41,95,1,73}已经连续存放在数

组 *data* 中(如图1所示)。

38	19	65	13	97	49	41	95	1	73	*key*
0	0	0	0	0	0	0	0	0	0	*file*

图1 数组 data

则精度归"档"插入排序过程如下:

a. 计算 $data[i].file$ 的值。归"档"系数 $L=97-1=96$, $data[i].file$ 由公式 $\lfloor(\frac{data[i].key-1}{L}+0.05)\times 10\rfloor$ 得到(如图2所示)。

38	19	65	13	97	49	41	95	1	73	*key*
4	2	7	1	10	5	4	10	0	8	*file*

图2 data [i]. file

b. 扫描数组 *data*,统计各"档"数据个数放入 *value* 中,其中第0、1、2、5、7、8"档"内包含数据元素1个,第4、10"档"内包含数据元素2个,第3、6、9"档"内包含数据元素0个,称之为空"档",并依此计算出所归各"档"首地址减1的值放入 *adr* 中,其中空"档"地址值与相继的非空"档"地址值相同(如图3所示)。

1	1	1	0	2	1	0	1	1	0	2

数组 *value*

0	1	2	3	3	5	6	6	7	8	8

数组 *adr*

图3 数组 *value* 和数组 *adr*

c. 扫描数组 *data*,将待排序数据依据数组 *adr* 中的地址值放入数组 *result* 中(如图4所示)。

第0"档"	第1"档"	第2"档"	第4"档"		第5"档"	第7"档"	第8"档"	第10"档"	
1	13	19	38	41	49	65	73	97	95

图4 数组 *result*

d. 将数组 *result* 中第4、10"档"进行局部的直接插入排序得到(如图5所示):

1	13	19	38	41	49	65	73	95	97

图5 已排好序的数组 *result*

3 精度归"档"插入排序算法分析

由精度归"档"插入排序算法基本思想不难看出,整个排序算法所耗费的时间 T 由6部分组成,其中 T_i 表示第 i 部分所耗费的时间:

1)初始化数组 *value* 和 *adr* 所耗费的时间 $T_1=O(22)$。

2)计算待排序数据最大值 *Max* 和最小值 *Min* 所耗费的时间 T_2,即使是在最坏情况下,也有 $T_2=O(n)$。

3)确定每个数据元素所在"档",并且统计各"档"内数据元素的个数所耗费的时间 T_3,因为这要扫描所有待排序数据一遍,故 $T_3=O(n)$。

4)确定各"档"首地址所耗费的时间 $T_4=O(10)$。

5)依据各"档"首地址将数据元素放入结果数组所耗费的时间 T_5,尽管 T_5 与待排序数据的分布有关,但由于每个数据至多被移动一次,因此有 $T_5=O(n)$。

6)对各"档"内数据元素进行直接插入排序所耗费的时间 T_6,以上几步除了 T_1、T_2、T_3 和 T_4 以外,只有 T_5 和 T_6 与待排序数据的分布有关,并且也只有 T_6 受到数据分布的影响。下面就从待排序数据均匀分布和非均匀分布两种情况来讨论 T_6,进而来讨论 T。

a. *待排序数据均匀分布* 根据精度归"档"插入排序算法可知:n 个待排序数据经过如上所述的归"档"计算以后,分布在11"档"中。这些"档"可能包含0个,1个,2个,…,m 个,甚至 n 个数据。由于在 n 个数据均匀分布的情况下,每一"档"落入数据的概率相同,都是 $1/n$,所以不难证明有 m 个数据落入同一"档"的概率为:

$$P_m=C_n^m(1/n)^m(1-1/n)^{n-m}=\frac{1}{m!}(1-1/n)^{n-m}\prod_{i=1}^{m-1}(1-i/n)\quad(m=1,2,\cdots,n)$$

因为 $(1-1/n)^{n-m}\prod_{i=1}^{m-1}(1-i/n)\leqslant 1$

故 $P_m<1/m!$

由于精度归"档"插入数据过程实际上是由对包含1个,2个,…,m 个,…,数据的那些"档"的插入组成,对于包含 m 个数据的一个"档",插入数据所需要的平均时间小于或等于 $C_2m^2+C_1m+C_0$(C_2,C_1,C_0 为常数),那么 n 个数据插入各"档"中排序所需要的平均时间(期望时间)为:

$$\begin{aligned}T_6&=\sum_{m=1}^{n}nP_m(C_2m^2+C_1m+C_0)<n\sum_{m=1}^{n}(1/m!)(C_2m^2+C_1m+C_0)\\&=n[\sum_{m=1}^{n}C_2m/(m-1)!+\sum_{m=1}^{n}C_1/(m-1)!+\sum_{m=1}^{n}C_0/m!]\\&=n[C_2(\sum_{m=1}^{n}1/(m-1)!+\sum_{m=2}^{n}1/(m-2)!)+C_1\sum_{m=1}^{n}1/(m-1)!+C_0\sum_{m=1}^{n}1/m!]\end{aligned}$$

因为 $\sum_{m=1}^{n}1/m!<e-1$

所以 $T_6<n(e-1)(2C_2+C_1+C_0)$

这表明,对于随机均匀分布的 n 个数据,在最坏情况下,各"档"插入排序之和 T 与数据 n 成线形关系,即 $T_6=O(n)$。由此可知 $T=T_1+T_2+T_3+T_4+T_5+T_6=O(n)$,综上所述,可以得到如下结论:在待排序数据均匀分布的情况下,精度归"档"插入排序的时间复杂度为 $O(n)$。

b. *待排序数据非均匀分布* 若待排序数据非均匀分布,但经过精度归"档"之后,落入各"档"内的数据却是均匀的,由 a 的讨论可知,该算法对此种非均匀分布的数据进行排序的平均时间复杂度为 $O(n)$。

精度归"档"插入排序算法,在"档"位已固定的基础上,利用归"档"统计和直接插入排序的特点,拓广了一般排序算法的适用范围,尤其对于分布不太好的记录,该算法的平均时间复杂度与数据量 n 成线形关系,明显优于其它常规算法。

4 实验结果

为了验证算法的效率,我们在微机上生成若干组随机数,调入相应的数组后,分别采用精度归"档"插入排序算法与冒

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