稳定快速排序算法研究

邵 顺 增
(常州工程职业技术学院计算机技术系 江苏 常州 213000)

摘 要 快速排序算法与其他算法相比是相当有效的排序算法,但此算法并不完善,它是不稳定的。为此,对快速排序算法进行改进,在每次对数据分割时,对需要移动的数据先分别顺序拷出并保存,分割结束前再按要求分别顺序拷入,使得新排序算法是稳定算法。理论分析和实验数据表明,在任何情况下,稳定快速排序算法都是稳定的,并且其他性能不比快速排序算法和归并算法差。

关键词 排序算法 算法稳定性 算法时间复杂度 算法空间复杂度 稳定快速排序

中图分类号 TP301.6 **文献标识码** A DOI:10.3969/j.issn.1000-386x.2014.07.067

STUDY ON STABLE AND QUICK SORT ALGORITHM

Shao Shunzeng
(*Department of Computer, Changzhou Institute of Engineering Technology, Changzhou 213000, Jiangsu, China*)

Abstract Quick sort algorithm is a more efficient one than other algorithms, but it is imperfect for its instability. Therefore, we improve the quick sort algorithm. At each time when the data is segmented, the new sort algorithm copies out and saves the data to be shifted in order separately, and then copies in them in order according to the need respectively before the segmentation is finished, that makes the new sort algorithm be the stable one. It is indicated that the stable and quick sort algorithm is stable in any case by the theoretical analysis and experimental data, and its performance is as good as the quick sort algorithm and the merge algorithm.

Keywords Sort algorithm Algorithm stability Algorithm time complexity Algorithm space complexity Stable and quick sort

0 引 言

快速排序算法在所有的排序算法中是比较优秀的算法,也是推荐的常用算法,但它不完善,即它是不稳定的排序算法。所谓排序算法的稳定性是指在待排序的记录序列中,如果存在多个具有相同的关键字的记录,若经过排序,这些记录的相对次序保持不变,即在原序列中,ri = rj,且 ri 在 rj 之前,而在排序后的序列中,ri 仍在 rj 之前,则称这种排序算法是稳定的,否则称为不稳定的。在实际处理数据的过程中,有时对排序的稳定性要求很高,例如在处理类似有奖问卷问题时,常根据问卷的成绩给部分人一定的奖品,但有时相同成绩的人很多,不可能都给奖品,一般采取的是在成绩相同的情况下,提交答案早的得奖品,提交答卷晚的不得奖品,这时就需要对成绩排序的算法是稳定的。

近年来,不少学者提出各种改进的快速排序算法,如三路基数快排[4]、高效快速排序算法[5]和超速快排[7]等,都是增强了快速排序时间性能,但对排序算法的稳定性基本上没有涉及。本文针对这一情况提出了一种稳定快速排序算法。

1 稳定快速排序算法设计

1.1 算法设计思想

经典的快速排序算法设计思想是通过一趟排序将待排记录分割成独立的二部分,通过直接交换方式使其中一部分记录的关键字均比另一部分记录的关键字小,然后再分别对这两部分的记录继续进行排序,以达到整个序列有序。在把待排序的数据分割成两部分时,需要把前后的数据直接进行交换,这样的交换可能导致数据前后颠倒,所以经典快速排序是不稳定的。要想使排序稳定,需要改掉这种直接交换前后数据的办法,采用其他方法处理。稳定快速排序算法就是这样一种改进算法。

稳定快速排序的设计思想如下:

(1) 通过一趟排序将要排序的数据划分成独立的两部分。

(2) 每部分各自再分成两小块,使一小块所有数据都按原顺序排列且比另一小块按原顺序排列的所有数据都小(或大)。

(3) 把四块中两个小(或大)块按原顺序连接起来,放在所有数据的前部,把另外两个不小于(或不大于)块也按原顺序连接起来,放在所有数据的后部。

(4) 然后再按此方法对形成的两部分数据分别进行稳定快速排序,直到所有数据都是有序的。整个排序过程可以递归进行。

1.2 算法设计思路

本文以递增顺序排序为例进行说明。

收稿日期:2012 - 12 - 16。江苏省"十二五"规划项目(D/2011/03/001,B - b/2011/03/003);2011 年常州工程职业技术学院基金项目(12JY010)。**邵顺增**,副教授,主研领域:管理信息系统分析与设计,TRIZ 创新教育。

（1）增加与待排序数据数量相同的辅助存储空间如图 1 所示。

图 1 辅助存储区

（2）将待排序序列按经典算法的处理方法相似的方法分成二部分如图 2 所示深色部分和浅色部分，同时把前面一部分数据进行如下处理，所有小于 middle 值的元素按原顺序存放在原来存储区最前部 D1 区，所有不小于 middle 值的元素按原顺序连续存放在辅助存储区最前部 T1 区；把后面一部分数据处理如下，所有大于等于 middle 值的元素按原顺序存放在原来存储区最后部 D4 区，所有小于 middle 值的元素按原顺序连续存放在辅助存储区最后部 T3。

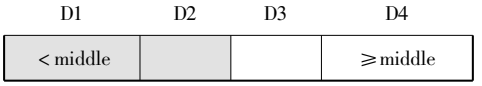

图 2 原数据分成的两部分

（3）把前面一部分数据中不小于 middle 的数据（T1 区数据）复制到后面一部分数据的最前部（D3 区），把后面一部分数据中小于 middle 的数据（T3 区数据）复制到前面一部分数据的最后部（D2 区）。一趟处理完成。

（4）再递归地对第一部分和第二部分分别进行上述操作，直至整个序列排序完成。

根据上面的算法设计思路可以知道，当被排序的数据中任何数据相等时，排在前面的数据永远排在前面，所以此算法是稳定的。

1.3 算法描述

由于使用已知数据和未知数据不影响排序算法描述，为了直观现以实际的数据为例对稳定快速排序算法进行描述，假设待排序的关键码序列为 Str1，排序前增加同样存储空间的整型数组 Str2，如图 3 所示。

Str1	34	21	53	8	78	123	21	53	34	111
Str2										

图 3 排序前的数据系列和需要的存储空间

第一趟稳定快速排序过程：

（1）选择 str1 中第一个数据 34 为 middle，作为数据分割依据。

（2）从 str1 前面依次判断每个数是否小于 middle，34 不满足小于 middle 条件，结束判断。

（3）从 str1 后面依次判断每个数是否大于或等于 middle，111 满足条件，34 满足条件，53 满足条件，21 不满足条件，结束判断。

（4）把 str1 前面的 34 移到 str2 的前头，把 str1 后面的 21 移到 str2 的后头，结果如图 4 所示。

Str1		21	53	8	78	123		53	34	111
Str2	34									21

图 4 第一次数据移动后的结果

（5）再回到 str1 前部继续判断 21，21 满足小于 middle 的条件，把 21 前移到原来 34 的位置，继续判断 53，53 不满足小于 middle 的条件，判断结束。结果如图 5 所示。

Str1	21		53	8	78	123		53	34	111
Str2	34									21

图 5 原数据区数据 21 前移结果

（6）再从 str1 后部继续判断，123 满足条件，把 123 后移到原来 21 的位置，78 满足条件，把 78 后移到原来 123 的位置，8 不满足条件，判断结束。结果如图 6 所示。

Str1	21		53	8		78	123	53	34	111
Str2	34									21

图 6 原数据区数据后移后结果

（7）把 str1 前部的 53 移到 str2 中 34 的后面，把 str1 后部的 8 移到 str2 中 21 的前面。至此第一轮判断结束，结果如图 7 所示。

Str1	21					78	123	53	34	111
Str2	34	53							8	21

图 7 一趟检查完结果

（8）最后把 str2 中前面的 34、53 顺序移到 str1 中 78 的前面，把 str2 中前面的 8、21 顺序移到 str1 中 21 的后面，结果如图 8 所示。到此整个第一趟稳定快速排序才真正结束。然后把分割的两部分再依次用上面方法完成整个稳定快速排序。

Str1	21	8	21	34	53	78	123	53	34	111
Str2										

图 8 第一趟分割后结果

1.4 算法实现

下面的代码为用递归思想描述的稳定快速排序算法（C 语言实现）。

```
int temp[Max];                    /*为存放需要移动数据存储空间*/
void stability_quick_sort (int * pData,long left,long right)
                                  /*稳定快速排序*/
{/*i,j 保存前后搜索的位置,i0,j0 保存原存储空间保留数据的位置,i1,j1 保存临时数据存储空间存储数据的位置*/
long i,j,i0,j0,i1,j1,i2,j2;
long middle;
i0 = i = i1 = left;
j0 = j = j1 = right;
middle = pData[left];
do{
   while((pData[i] < middle) && (i <= j))
                                  /*从左扫描小于 middle 的数*/
   {
      if(i0 != i)
      {
        pData[i0] = pData[i];
      }
      i0 ++;                      /*改变位置*/
      i ++;
   }
   while((pData[j] >= middle) && (j >= i))
                                  /*从右扫描大于等于 middle 的数*/
   {
      if(j0 != j)
      {
```

```
            pData[j0] = pData[j];
        }
        j0 --;
        j --;
    }
    if(i < j)                                   /* 找到了一对数据 */
    {
/* 把不满足条件的数据分别移到临时存储空间的前面和后面 */
        temp[i1] = pData[i];
        temp[j1] = pData[j];
        j1 --;                                  /* 改变位置 */
        i1 ++;
        i ++;
        j --;
    }
    else if(i == j)
    {
        i ++;
        j --;
    }
    else
    {
        i ++;
    }
}while(i <= j);

/* 把 temp 中前面的数据复制到 pData 中后部数据的前面 */
for(i2 = i1 - 1; i2 >= left; i2 --, j0 --)
    pData[j0] = temp[i2];
/* 把 temp 中后面的数据复制到 pData 中前部数据的后面 */
for(j2 = j1 + 1; j2 <= right; j2 ++, i0 ++)
    pData[i0] = temp[j2];
if(left < j)
stability_quick_sort (pData, left, j);
if(right > i)
stability_quick_sort (pData, i, right);
}
```

2　稳定快速排序算法理论分析

本稳定快速排序算法是在原来的快速排序算法的基础上改进的,算法理论与原快速排序的算法理论相似,在此只对改进的地方进行分析。

2.1　算法稳定性

本稳定快速算法在进行数据分割的过程中,把需要前后移动的数据不是直接交换,而是先把它们按原顺序移出来保存起来,被移走数据的位置用后面(或前面)的数据前移(或后移)填充,等所有数据检查、移出和前后移完成后,把保存起来要移动的数据分别移回到原来存储区前面所有数据的后面和后面所有数据的前面,完成一次数据分割。然后把分割结果的前后两部分数据再按前述方法进行分割,直到所有的数据都被分割成单个数据,完成排序。确保相同的数据按原来的顺序存放即排序的稳定性。

2.2　算法空间复杂度

稳定快速排序算法与原来的快速排序算法相比,在空间上多用了与原排序数据相同数据量的排序空间,用来存放需要前后移动的数据,它的空间复杂度还是 O(N)。

2.3　算法时间复杂度

稳定快速排序算法与经典快速排序算法在处理方式方面的主要不同在于对需要移动的数据处理方式有所差异,算法时间复杂度不同就是由此引起的。

经典快速排序算法若需要数据移动,通过前后两数据交换完成,需要下面的处理过程:

```
Temp = pData[i];
pData[i] = pData[j];
pData[j] = temp;
```

稳定快速排序算法若需要数据移动,先把数据顺序存放到临时存储区:

```
Temp[i1] = pData[i];
Temp[j1] = pData[j];
```

当所有数据检查处理完后,再把临时存储区中所有数据顺序移回原数据区:

```
pData[i0] = temp[j2];
pData[j0] = temp[i2];
```

综上所述,改进算法与经典算法相比每次移动数据只是多了一个赋值语句,其他语句基本相同,因此对 n 个排序数排序时最坏情况需移动 n 个数,假定花费时间为 Tf(n),对 n 个排序数进行一次划分的时间代价为 C(n),则:

稳定快速排序的总时间代价为:

$$
\begin{aligned}
T(n) = Tf(n) + C(n) &= Tf(n) + n + 2C(n/2) \\
&= Tf(n) + n + 2[Tf(n/2) + n/2 + 2C(n/4)] \\
&= 2Tf(n) + 2n + 4C(/4) = \cdots \\
&= kTf(n) + kn + 2kC(n/2k) = \cdots \\
&= \log(n) * [Tf(n) + n] + nC(1) \\
&= \log(n) * [Tf(n) + n] = O(n\log(n))
\end{aligned}
$$

故,稳定快速排序的算法没有引起时间性能大的变化,算法时间复杂度仍为 O(NlogN)。

3　实验结果与结论

3.1　稳定性测试

为了验证稳定快速算法,随机组合了大量数据进行了实验,如使用如图 9 表中十条记录的数据用本算法进行了简单的测试,根据字段 1 按递增排序,排序后记录数据的输出效果如图 10 所示。

字段 1	字段 2
2	0
2	1
11	12
2	2
2	3
56	1
86	2
85	4
2	4
10	2

图 9　排序的数据记录

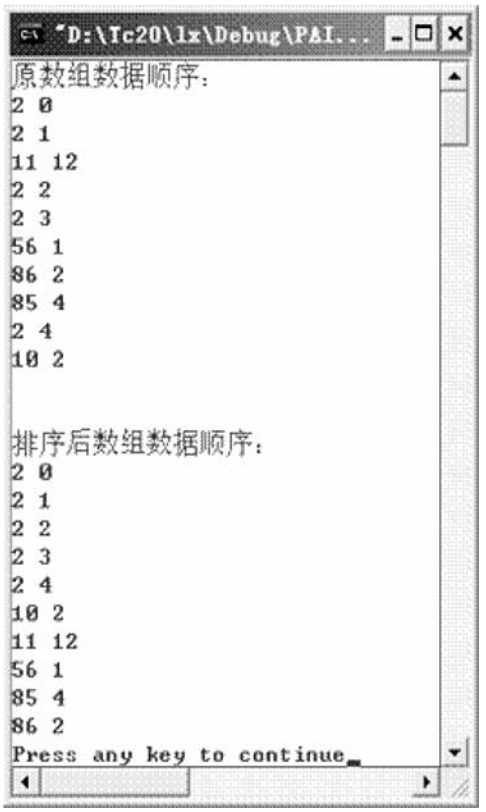

图 10 对十条记录排序后结果

从图中的结果可以看出,根据字段 1 排序后,字段 1 中相同的数据的先后顺序没有发生改变,字段 2 为 0 的排第一位,字段 2 为 1 的排序第二位,字段 2 为 4 的排第五位,因此算法是稳定的。

3.2 时间复杂度测试

在 Windows XP 的 VC 6.0 平台上,对大量数据进行多次测试(数据由随机函数 Rand()产生)。

1) 随机产生 50 000 个整型数据用各种排序方法排序所用时间基本如图 11 所示。由图 11 可知,稳定快速排序有很好的性能表现,速度与原快速排序基本上相同,与稳定排序的归并排序也基本相同。

图 11 对 50 000 个整型数据排序相对时间结果

2) 随机产生 50 000 个整型数据,然后用求余法得到 50 000 个 1 位整数、50 000 个 2 位整数、50 000 个 3 位整数、50 000 个 4 位整数和 50 000 个 5 位以上整数用稳定快速排序法进行排序所用时间基本如图 12 所示。由图 12 可知,当数据的位数少时,重复的数据多,需要移动的数据多,排序效率较慢,3 位以上数据的排序效率基本相同,达到了最好。

图 12 对不同位数排序相对时间结果

4 结 语

稳定快速排序算法是在原快速排序算法的基础上,对算法的稳定性进行的完善,理论和实践证明,使用稳定快速排序算法,可以确保排序是稳定的,且本算法的时间复杂度没有大的改变,只是多用了与排序数据数量相同的存储空间。

参 考 文 献

[1] 庞建雄. 排序算法稳定性的深入讨论[J]. 桂林电子工业学院学报,1996,16(1):1-4.
[2] 汪沁,奚李峰. 数据结构[M]. 北京: 清华大学出版社,2009.
[3] 秦锋. 数据结构[M]. 合肥: 中国科学技术大学出版社,2007.
[4] 王善坤,陶祯蓉. 一种三路划分快速排序的改进算法[J]. 计算机应用研究,2011,29(7):2513-2516.
[5] 汤亚玲,秦锋. 高效快速排序算法研究[J]. 计算机工程,2010,36(7):77-78.
[6] Cantone D, Cincotti G. Quic kHeapsort: An Efficient Mix of Classical Sorting Algorithms[J]. The Oretical Computer Science, 2002, 285: 25-42.
[7] 周建钦. 超快速排序[J]. 计算机工程与应用,2006,42(29):41-43.
[8] 郭晶旭. 基于快速排序的改进算法[J]. 计算机科学,2006,39(4A):343-344.

(上接第 254 页)

[6] 刘磊,闫德勤,桑雨. 连续属性离散化的 Bayesian-Chi2 算法[J]. 计算机工程与应用,2008,44(18):39-41.
[7] Kerber R C. Discretization of Numeric Attributes[C]//Proceedings of the 10th National Conference on Artificial Intelligence. MIT Press, 1992:123-128.
[8] 李刚,李霁伦. WILD:基于加权信息损耗的离散化算法[J]. 南京大学学报:自然科学版,2001,37(2):148-152.
[9] Xia Yongxiang, Shi Zhicai. An incremental SVM for intrusion detection based on key feature selection[C]//Proceedings of the Third International Symposium on Intelligent Information Technology Application, 2009:205-208.