一种三路划分快速排序的改进算法

王善坤[1], 陶祯蓉[2]
(1. 大连理工大学 城市学院 网络信息中心, 辽宁 大连 116600; 2. 四川省计算机研究院, 成都 610041)

摘 要: 快速排序是一种经典的排序算法,它的平均性能非常突出。针对快速排序在某些特殊情况下(如数据已有序或重复数据较多时)效率较低的问题进行了研究,对三路快速排序进行改进,使快速排序在特殊情况下也能保持较好的效率。通过大量的数据测试发现,该算法在最好情况下其性能在几个数量级上优于普通快速排序,在最坏情况下,其性能较普通快速排序无明显差距。改进后的三路快速排序是一种通用高效的排序算法,因此在某些情况下选用、该算法会获得更好的效率。

关键词: 快速排序; 平均时间复杂度; 三路划分快速排序; 算法; 排序性能
中图分类号: TP301.6 **文献标志码**: A **文章编号**: 1001-3695(2012)07-2513-04
doi:10.3969/j.issn.1001-3695.2012.07.029

Enhanced algorithm for three-route quick sort

WANG Shan-kun[1], TAO Zhen-rong[2]
(1. *Network Information Center, City Institute, Dalian University of Technology, Dalian Liaoning* 116600, *China*; 2. *Sichuan Institute of Computer Sciences, Chengdu* 610041, *China*)

Abstract: Quick sort is a kind of classic sorting method whose average operation stands out. For the low efficiency problem of the quick sort in some special cases(when dealing with ordered or repetitive data), the algorithm improved the three-way quick sort, so that in special cases, the algorithm still maintainsed good efficiency. Large number of tests show that, in its best scenario, this calculating approach is largely superior to the ordinary ones, and in its worst scenario, it equals to the ordinary ones. The improved three-way quick sort is a general and efficient sorting algorithms ,so in certain case, it may provide access to more efficiency.

Key words: quick sort; average time complexity; three-route quick sort; algorithm; efficiency for sorting

0 引言

快速排序是一种经典的排序方法,其思想是在待排序的序列$\{v[s],v[s+1],\cdots,v[t]\}$中,首先选取一个元素(通常选取第一个元素$v[s]$)作为枢轴,然后按下述原则重新排列其余记录:将所有值小于它的元素都安置在其位置之前,将所有值大于它的记录都安置在其位置之后;以此枢轴元素最后的位置i作为分界线,将序列分割为二个子序列,再对每个子序列重复上面的过程,直到整个序列排序好为止。对上述序列进行第一次分割后,其状态如图 1 所示。

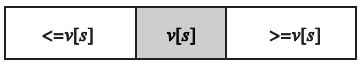

图 1 普通快速排序示意图

其中:$<=v[s]$表示所有值小于或等于$v[s]$的元素,$>=v[s]$表示所有值大于或等于$v[s]$的元素。

1 现有研究成果及比较

基本快速排序是一种高效的排序方法,近年来被国内学者从各种不同的角度进行了改进与优化。

宋鸿陟等人[1,2]从多线程编程的角度对基本快速排序进行改进,优点是在多核处理器平台下,其算法会充分利用处理器,提高排序效率;在单核处理器平台下或在主频较低的多核处理器下,其效率反而不如一般的快速度排序,因为在这种情况下,其算法自然退化为基本快速排序,此外还要附加对线程处理的负担。所以其算法的本质是针对多核处理器进行优化。

庹清[3]从混合快速排序与其他排序的角度进行优化,其算法思想是将基本快速排序与计数排序混合,从而构造高效的排序方法。其算法效率远高于一般快速排序,缺点是在降低算法时间复杂度的同时提高了空间复杂度,其算法的本质是用更多的空间来换取更快的速度。

陈宝平[4]则是针对特定类型的待排序数据对基本快速排序进行改进,其算法的本质与三路划分快速排序基本无异。

汤亚玲[5]从概率学的角度,先对待排序数据进行统计并选取出最佳的枢轴元素,以确保快速排序的每一次划分位置都正好处于待排序序列的正中间。其算法的本质是先选取合适的枢轴,再进行基本快速排序,而选取合适的枢轴本身就是一个很浪费时间的操作,因此其方法只能在某些特定情况下提高排序效率,而在另一些情况下反而效率低于基本快速排序。

本文给出的改进方法与几位同仁的方法有所不同,并不是针对某种特定情况对算法进行改进,而是假定待排序的数据是未知的,处理器的内核数也是未知的,使改进后的算法具有一

收稿日期: 2011-12-01; **修回日期**: 2012-02-10

作者简介:王善坤(1960-),男,讲师,学士,主要研究方向为数据库应用、计算机网络、嵌入式应用等(wangsk@dlut.edu.cn);陶祯蓉(1961-),女,高级工程师,主要研究方向为数据库、网络信息管理系统、计算机应用等.

定的通用性,同时也对特殊数据(如正序、逆序)进行了兼顾。该算法对通用数据的排序效率不低于基本快速排序,而对某些特殊数据,其效率远远高于基本快速排序。

2 三路划分快速排序

当待排序的元素序列中有大量的重复码时,前面讲的经典快速排序效率将会降得很低,三路划分的快速排序可以改进这方面的不足。三路划分排序的思想是将待排序序列分为三部分,最前面是所有小于枢轴元素值的元素;中间是等于枢轴元素值的元素,最后面是所有大于枢轴元素值的元素。如果枢轴元素的值为 $v[s]$,图 2 很好地说明了对序列的划分。

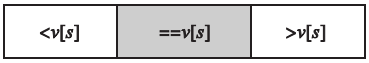

图2 三路划分快速排序示意图

其中:$<v[s]$表示所有值小于 $v[s]$ 的元素;$>v[s]$表示所有值大于 $v[s]$的元素;$==v[s]$表示所有值等于 $v[s]$的元素。

进行完上面的过程后,中间部分数据的位置就确定下来了,然后再递归地对第一部分和第三部分进行上述操作,直至整个序列排序完成。

3 加强型三路划分快速排序算法设计

3.1 算法设计思路

根据上面的描述可以知道,当被排序的数据中任何两个数据都不重复时,三路快速排序蜕化为经典快速排序,并有额外开销,因此在这种情况下三路快速排序的性能不如经典快速排序,表 1 中的 14 ~ 15 行也表明事实确实如此。反过来讲,经典快速排序也是一种三路快速排序,只不过是中间的那一路只含有一个元素。于是考虑能否在没有重复数据的情况下也使中间的那路不为空,解决的办法是中间的那路数据不是值等于某个具体值的数据,而是在某个范围的数据,这样即使待排序的数据没有重复值,被划分出的中间那路也可以有不止一个元素。因此在本算法中需要有两个枢轴元素,在这里分别记为 v_1、v_2。图 3 可以很好地表示进行一次划分后序列的状态。

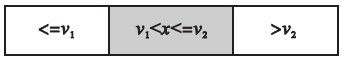

图3 加强型三路划分快速排序示意图

其中:$<=v_1$ 表示所有值小于或等于 v_1 的元素;$>v_2$ 表示所有值大于 v_2 的元素;$v_1<x<=v_2$ 表示所有值大于 v_1 且小于或等于 v_2 的元素。

3.2 算法描述

a)设待排序的数据存在数组 $v[\,]$中,数组的上、下界分别为 low 与 high。

b)若待排序元素个数少于三个,使用直接插入排序。

c)将 low 的值保存到变量 originLow 中,即 originLow = low;将 high 的值保存到变量 originHigh 中,即 originHigh = high。

d)取 $v_1 = v[\text{low}]$,$v_2 = v[\text{high}]$,若 $v_1 == v_2$,则重复执行 low = low + 1,$v_1 = v[\text{low}]$,直至 $v_1 \neq v_2$ 或 low == high。

e)若执行完上面的代码后 low == high,则说明待排序的数据全相等,程序结束;否则执行 f)。

f)如果 $v_1 > v_2$,则交换 v_1 与 v_2 的值。

g)从 originLow 位置开始,从前向后扫描数组,找到第一个值大于 v_1 的元素,将其下标记为 i,并将 i 的值赋给 current。

h)从 originHigh 位置开始,从后向前扫描数组,找到第一个值小于或等于 v_2 的元素,将其下标记为 j。

i)当 current <= j 时,重复进行下面的操作:

(a)比较 $v[\text{current}]$与 v_1 的值,若前者小于等于后者,则交换 $v[\text{current}]$与 $v[i]$的值,并执行 current = current + 1,$i = i + 1$;否则转(b)。

(b)比较 $v[\text{current}]$与 v_2 的值,若前者大于后者,则交换 $v[\text{current}]$与 $v[j]$的值,并执行 $j = j - 1$。

(c)若(a)与(b)中的条件都不满足,则执行 current = current + 1。

3.3 算法实现

代码 1 加强的三路划分快速排序

```
1   void ProThrQSort(int v[], int low, int high)
2   {
3   int originLow = low, originHigh = high;
4   int i = low, j = high, current;
5   if((high - low) < 2)
6   {
7       DirectInsertionSort(v, low, high);
8       return;
9   }
10  int v1 = v[low], v2 = v[high];
11  while(v1 == v2)
12  {
13      v1 = v[++low];
14      if(low == high) return;
15  }
16  if(v1 > v2) swap(v1, v2);
17  while(v[i] <= v1) i++;
18  while(v[j] > v2) j--;
19  current = i;
20  while(current <= j)
21  {
22      if(v[current] <= v1) swap(v[current++],
        v[i++]);
23      else if(v[current] > v2) swap(v[current], v[j--]);
24      else current++;
25  }
26  ProThrQSort(v, originLow, i-1);
27  ProThrQSort(v, i, current-1);
28  ProThrQSort(v, current, originHigh);
29  }
```

3.4 关键代码注解

7 行:直接插入排序中有越界判断,即如果 low >= high,则函数不做任何事情,直接返回。

10 行:在程序中,需要找到两个值不相等的数,分别为 v_1、v_2,在此将 v_1 设为待排序元素中的第一个元素,将 v_2 设为待排序元素中最后一个元素。

11 ~ 15 行:此段代码的功能是为了防止 v_1 与 v_2 的值相等。如果 v_1 与 v_2 的值相等,则从前向后扫描,直到找到第一个不与 v_2 相等的数,并用 low 代表这个数的下标。若从前向后扫描完毕后,发现待排序的所有元素的值都相等则程序结束,否

则必有 $v_1 \neq v_2$。

16 行:为了表述方便,使 v_1 成为 v_1 与 v_2 中值较小的数,v_2 成为 v_1 与 v_2 中值较大的数。

17 ~ 25 行:此段代码用来将待排序的序列划分为三路:第一路即最左边的一路,该路中所有元素的值都小于或等于 v_1;第二路即中间的一路,该路所有元素的值都大于 v_1 并且小于或等于 v_2;第三路即最右边的一路,该路中所有元素的值都大于 v_2。上述三路中的每一路元素皆无序。

17 行:当该行代码运行结束时,位置位于 i 之前的元素皆属于第一路。

18 行:当该行代码运行结束时,位置位于 j 之后的元素皆属于第三路。

19 ~ 25 行:此段代码筛选位置位于 current 与 high 之间的每一个元素,并将某些元素插入到第一路或第三路。当筛选结束后,未被插入到第一路或第三路的元素即是中路元素。

22 行:在 if(v[current] <= v_1) 中,v[current]代表当前被筛选的元素。当 v[current]小于或等于 v_1 时,说明 v[current]应该属于第一路,而 i 既代表中路第一个元素的位置,又代表第一路最后一个元素下一个元素的位置,所以将其与 $v[i]$ 交换。交换后,current 位置的元素本身就是中路的元素,所以下标 current 的值需要加 1 以避免重复筛选。又由于向第一路中新插入了一个元素,所以 i 的值需要相应地加 1。

23 行:如果 v[current]的值大于 v_2,那么 v[current]应该被移至第三路,j 代表当前第三路第一个元素之前一个元素的下标,所以将 v[current]与 $v[j]$ 交换,交换后 current 处的元素相当于还未经筛选,所以 current 的值不变,而 j 值减 1。

4　时间复杂度分析

在最好的情况下,即所有元素值都相等的情况下,只需将元素从前向后扫描一遍程序就结束,其时间复杂度明显为 $O(n^2)$。最坏的情况即为数据逆序或正序,这时 $T(n) = C_1 n + T(n-1) = C_1 n + C_2(n-1) + T(n-2) = \cdots = O(n^2)$。在最理想的平均情况下,即每一次划分都恰好将原序列划分为三个等长的序列,其时间复杂度为

$$T(n) = C_1 n + 3T(n/3) = C_1 n + 3(C_2 \times n/3 + 3\ T(n/9)) = C_1 n + C_2 n + 9T(n/9) = \cdots = O(n\log_3 n)$$

5　测试

5.1　测试数据

测试数据共 15 组,每组由 40 万非负整数组成。每组测试数据中整数的取值范围不同。

5.2　测试数据的生成

以其中一组为例,设该组由 40 万非负整数组成,且每个整数的值为 0 ~ 9999 间的随机整数。产生其中一个随机整数的方法为:先产生一个长度为 10 000 的数组 int v[10000];然后为数组元素附值 $v[i] = i$,即 $v[0] = 0$, $v[1] = 1$, $v[2] = 2$,…, $v[9999] = 9999$,最后调用 C ++ 标准模板库中的乱序函数将数组中元素乱序排序,乱序后 $v[0]$ 即是所要的整数。将乱序函数重复调用 40 万次,就得到 40 万个等概率随机整数,每个整数的取值范围为 0 ~ 9999。

5.3　随机数产生代码

```
1  int GenerateRandomNum( int n )
2  {
3   std::vector<int> v1( n );
4   for ( int i=0; i<n; ++i )
5   {
6   v1[i] = i;
7   }
8   std::random_shuffle(v1.begin(),v1.end());
9   return (*v1.begin());
10  }
```

5.4　测试过程

分别调用经典快速排序、三路划分快速排序和加强型三路划分快速排序对每一组数据进行排序,记录每一次排序所用的时间并且列表汇总。

5.5　计时方法

在排序开始前和结束后分别调用 C/C ++ 中的计时函数 clock(),并用二次函数返回值的差除以 1 s 内的时钟计时单元个数 CLOCKS_PER_SEC,就得到了排序函数运行的时间。

5.6　计时代码

```
1  start = clock();
2  ThreeDivQuickSort(v, 0, n - 1);
3  finish = clock();
4  duration = 1000 * (double)(finish - start) / CLOCKS_PER_SEC;
5  cout << duration << "ms" << endl;
```

5.7　测试平台

处理器 :AMD B45

操作系统:Windows 7

编译器:Visual C ++ 6.0

5.8　测试结果(表 1)

表格 1　测试结果

	数据文件(每文件中含 40 万个正整数)	文件中任一整数 X 的范围	普通快排/ms	三路快排/ms	加强三路快排/ms
1	c:\1.txt	$X=1$	287 699	16	2
2	c:\10.txt	$0 \leqslant X \leqslant 9$	24 012	23	15
3	c:\100.txt	$0 \leqslant X \leqslant 99$	2 362	36	32
4	c:\500.txt	$0 \leqslant X \leqslant 499$	491	48	50
5	c:\1000.txt	$0 \leqslant X \leqslant 9\ 999$	273	48	53
6	c:\5000.txt	$0 \leqslant X \leqslant 4\ 999$	99	60	65
7	c:\10000.txt	$0 \leqslant X \leqslant 9\ 999$	78	63	69
8	c:\20000.txt	$0 \leqslant X \leqslant 19\ 999$	69	68	76
9	c:\50000.txt	$0 \leqslant X \leqslant 49\ 999$	65	70	78
10	c:\100000.txt	$0 \leqslant X \leqslant 99\ 999$	64	72	83
11	c:\200000.txt	$0 \leqslant X \leqslant 199999$	64	74	88
12	c:\300000.txt	$0 \leqslant X \leqslant 299\ 999$	63	73	86
13	c:\400000.txt	$0 \leqslant X \leqslant 399\ 999$	64	76	84
14	c:\sort.txt	0 ~ 399999 正序	287 655	383 808	87 725
15	c:\rsort.txt	0 ~ 399999 逆序	277 354	377 095	87 744

5.9　测试结果分析

对表 1 的分析如下:

a)第1行显示当待排序的数据为等值时,三路划分快速排序和加强型三路划分快速排序的性能好于经典快速排序好几个数量级,加强型三路划分快速排序优于三路划分快速排序。

b)2~5行显示待排序的数据在重复率较高时,三路划分快速排序和加强型三路划分快速排序的时间性能远好于经典快速排序。

c)6~7行显示待排序的数据在重复率略高时,三路划分快速排序和加强型三路划分快速排序略优于经典快速排序,但优势不明显。

d)8~13行显示待排序的数据在重复率较低时,经典快速排序略优于三路划分快速排序和加强型三路划分快速排序,但优势不明显。

e)14~15行显示在正序和逆序这两种特殊情况下,加强型三路划分快速排序的效率远好于经典快速排序和三路划分快速排序。

6 结束语

表1显示在一般情况下加强型三路划分快速排序的效率与三路划分快速排序效率基本一致,且待排序的数据在重复率较高时,加强型三路划分快速排序效率略优于三路划分快速排序;待排序的数据在重复率较低时,三路划分快速排序略优于加强型三路划分快速排序。在特殊情况下(所有数据等值、正序、逆序三种情况),加强型三路划分快速排序远远优于三路划分快速排序。

从表1中还可以得出一个结论:加强型三路划分快速排序的效率与待排序的数据的重复率成正比,当所有数据都是同一个数据时,这时重复率最大,加强型三路划分快速排序的效率也最好。

因此,在事先能预测到待排序数据的数据重复率较高时,可以优先考虑使用加强型三路划分快速排序。

参考文献:

[1] 宋鸿陟,傅熠,肖磊,等.归并方式的多线程快速排序算法[J].计算机教育,2010(8):149-152.

[2] 宋鸿陟.分割方式的多线程快速排序算法[J].计算机应用,2010,30(9):2374-2378.

[3] 庹清.改进的按位拆分快速排序算法[J].计算机应用,2011,31(1):55-57.

[4] 陈宝平.高重复率数据的快速排序[J].电子科技,2011,24(8):22-23,30.

[5] 汤亚玲.高效快速排序算法研究[J].计算机工程,2011,37(6):173-175.

[6] 严蔚敏.数据结构(C语言版)[M].北京:清华大学出版社,2010:272-277.

[7] 殷人昆.数据结构(用面向对象方法与C++语言描述)[M].2版.北京:清华大学出版社,2007:405-413.

[8] 郭晶旭.基于快速排序的改进算法[J].计算机科学,2009,36(4A):343-344.

[9] 周建钦.超快速排序算法[J].计算机工程与应用,2006,42(29):41-42,86.

[10] POWERS D M W. Parallelized quick sort with optimal speedup [EB/OL]. [2010-01-11]. http://www.cs.ucsb.edu/~gilbert/cs140Win2009/sortproject/ParallelQuicksort.pdf.

[11] CEDERMAN D,TSIGAS P. A practical quick sort algorithm for graphics processors[J]. Journal of Experimental Algorithmics,2009,14(4):4-24.