浅析 C 语言快速排序算法的改进

Analysis of improving the c language quick sort algorithm

刘 娜 （渤海大学信息中心 辽宁锦州 121013）
佟 冶 （渤海大学文理学院 辽宁锦州 121013）

摘 要： 排序是计算机程序设计中一种重要操作，本文论述了 C 语言中快速排序算法的改进，即快速排序与直接插入排序算法相结合的实现过程。在 C 语言程序设计中，实现大量的内部排序应用时，所寻求的目的就是找到一个简单、有效、快捷的算法。本文着重阐述快速排序的改进与提高过程，从基本的性能特征到基本的算法改进，通过不断的分析，实验，最后得出最佳的改进算法。

关键词： 算法 改进 插入式排序 快速排序

1 引言

对于一个好的排序算法，应该是在不断总结和改进的基础之上得出的，快速型排序是比其他任何算法都应用广泛的排序算法，在排序算法中是最流行、最实用的。其原因是它易于实现，能够很好地适用于不同的输入数据，而且在很多场合它消耗的资源比其他任何排序方法都少。在计算机上实现时，一个精心编制的快速排序版本比其他任何排序方法要快很多，快速排序广泛被用作库排序工具，而且还被应用于其他意义重大的应用中。

2 基本方法的准备与实现

在全面涉及内部排序算法时，首先要掌握几种适合小文件，思维过程易理解的基本排序算法。借助这些算法，我们可以掌握排序算法的一些基本术语和基本原理。快速排序是对冒泡排序的一种本质的改进。它的基本思想是通过一趟扫视后，使待排序的长度能较大幅度地减少。在冒泡排序中，一次扫视只能确保最大键值的结点移动到了正确位置，而待排序序列的长度可能只减少 1。快速排序通过一次扫视使某个结点移动到了正确位置，并使他在它左边序列的结点的键值都比它的小，而它右边序列的结点的键值都不比它的小。我们称这样一次扫视为“划分”。每次划分使一个长序列变成新的较小子序列，对这两个小子序列分别作同样的划分，直到新的子序列的长度为 1 时才不再划分。当所有的子序列长度都为 1 时，序列便已排好。

在已有的快速排序基础之上，我们可以对快速排序进行改进，目的是寻找一个更合适的中间元素，来对文件进行高效率的划分，一个寻求较好的划分元素的方式是从文件中取出 3 个元素的样本，然后使用这 3 个元素的中值作为划分元素。分别从数组的左、中、右边选取一个元素，对这 3 个元素排序。把中间元素作为第一个元素，左边的作为第二个，右边的作为第三个。即保证了左边的一个为中值，右边的最大，避免了一些交换。利用此方法可以在递归基础上作改进，进而得出三元素中值法划分的快速排序。

基本算法如下：

函数：quicksort(int e[ ],int low,int high)

形参说明：e 为接收待排序数组的首地址，low 为待排序数组的起始下标，high 为待排序数组的末下标。

```
exchange( int e[ ],int low,int high)
{
  /* 对三个数进行确定中值，待选的三个数为数组中左中右三个数，即实现交换 */
  int t,i = ( low + high )/2; /* 确定中间数的下标 */
  if( e[i] > e[low]) /* 通过三个单分支实现中值落在下标为 low 的位置上 */
  {t = e[i];e[i] = e[low];e[low] = t;}
  if( e[i] > e[high])
  {t = e[i];e[i] = e[high];e[high] = t;}
```

```
    if(e[low] > e[high]) /* 最后一个分支进行定位中值 */
    {t=e[low];e[low]=e[high];e[high]=t;}
  }
  quicksort(int e[],int low,int high)
  { int i,j,t;
    if(high - low >=3) /* 有必要进行确定中值 */
    exchange(e,low,high); /* 调用交换函数 */
    if(low < high) /* 对 low 至 high 以 e[low]为基准作划分 */
    {i=low;j=high;t=e[low]; /* 用基本快速排序方法实现确定位置 */
    while(i<j)
     { while(i<j&&e[j]>t) j--;
     if(i<j) e[i++]=e[j];
     while(i<j&&e[i]<=t) i++;
     if(i<j) e[j--]=e[i];
      }
    e[i]=t;
    quicksort(e,low,i-1); /* 递归,对左子序列作划分 */
    quicksort(e,i+1,high); /* 递归,对右子序列作划分 */
    }
  }
```

3 分析与改进

三元素中值法在三个方面对快速排序有帮助。首先,它使最坏情况在实际排序中出现的概率更小。因为检查的元素必定处于文件的最大与最小元素之间,而且这种情况在大部分划分中始终如此。其次,它避免了划分中对标记键的需求,因为这个函数由这三个元素中之一来担任,他们在划分前就进行了检查。第三,它减少了算法的总平均运行时间。在处理问题的时候,我们采用的是递归的办法,程序虽然利用的是三元素中值法,但在递归快要结束的时候,中值法进行交换时的意义不大,所以可以将其改进。如果数组的长度小于M(可令M=10)时就停止递归。这样的快速排序结束后没有达到预期的目的,但它具有一定的初始序列,我们可以将其进行一下直接插入排序,完成了本次改进。具体函数程序如下:

函数:void sort(int a[],int l,int r)

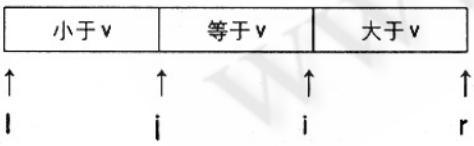
形参说明:a 为接收待排序数组的首地址,l 为数组的起始下标,r 为数组的末下标。

```
void quicksort(int a[],int l,int r)
{ int i,j,t,e,v;
  if(r-l<=M) return; /* 如果长度小于 M ,停止递归 */
  t=a[(l+r)/2]; a[(l+r)/2]=a[l+1]; a[l+1]=t; /* 将中间值同第二个元素交换, */
  if(a[l]>a[l+1]) {t=a[l]; a[l]=a[l+1]; a[l+1]=t;}
  if(a[l]>a[r]) {t=a[l]; a[l]=a[r]; a[r]=t;}
  if(a[l+1]>a[r]) {t=a[l+1]; a[l+1]=a[r]; a[r]=t;}
    /* 上面三行将三个元素(第一个,第二个和最后一个)进行由小到大排序 */
  i=l+1; j=r-1; /* i 下标为中值 */
  v=a[i]; /* 将中值存储在 v 中 */
  i++;
  for(;;)
  { while(a[++i] < v); /* 在左端检测,如果遇到比 v 大的停止 */
while(v < a[--j]) if(i>=j) break; /* 如果遇到比 v 小的停止 */
if(i>=j) break;
t=a[i]; /* 交换检测到的两个值 */
a[i]=a[j];
a[j]=t;
}
t=a[i]; /* 将中值交换到恰当的位置 */
a[l]=a[l+1];
a[l+1]=t;
quicksort(a,l+1,i-1); /* 递归,对左子序列作划分 */
quicksort(a,i+1,r-1); /* 递归,对右子序列作划分 */
  }
   insertion(int a[],int l,int r) /* 直接插入排序函数 */
```

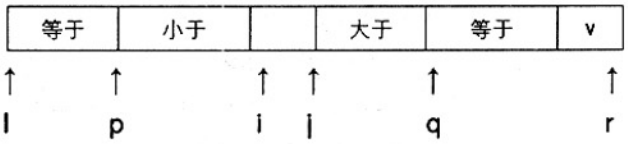
```
{int i,j,t;
for(i=l+1;i<=r;i++)
{t=a[i];
j=i-1;
while(a[j]>t &&j>=l)
{a[j+1]=a[j];
j--;
}
a[j+1]=t;
}
}
void sort(int a[],int l,int r) /*可认为是一个三元素中值法快速排序函数*/
{ quicksort(a,l,r); /*先中值法,得到一个接近有序的序列*/
insertion(a,l,r); /*利用直接排序法得到完整的序列*/
}
```

4 特殊问题的应用

在排序应用过程中,经常出现具有大量重复键的文件。例如,按出生年份排序一个职员文件,或者使用排序来区分女性与男性,当存在这样许多重复键时,快速排序性能会变得低下,让人无法接受,但可以进行实质的改变。一个直接的想法是,将文件分成三部分,每部分分别对应小于划分元素的键、等于划分元素的键以及大于划分元素的键,进而可以得出三路划分的快速排序。如下所示:



完成这个划分要比二路划分更复杂,有许多不同的方法可用于完成此项任务。Bentley 和 McIlroy 在 1993 年发明的一个解决三路划分的聪明方法是,修改标准划分方案如下:将左边子文件中碰到的与划分元素相等的键放到文件的左端,将右边子文件中碰到的与划分元素相等的键放到文件的右端。在划分过程中,应该保持如下情形:



然后,当指针交叉而且相等键的准确位置已知时,将所有与划分元素相等的键的项交换到位。这种方案不是很符合三路划分在文件的一次遍历中完成要求,但是重复键的额外开销只与发现的重复键数量成比例。这表明两点:一是,即使没有重复键,这种方法也非常有效,因为不存在额外的开销。二是,当仅存在常数个键值时,该方法为线性时间:每个划分阶段从排序中排序所有具有与划分元素相等的值的键,所以,每个键最多在一定数量的划分中用到。

具体利用本思想的三路划分快速排序是将数组划分为三部分:小于划分元素的元素(放在 a[l],……,a[j]);等于划分元素的元素(放在 a[j+1],……,a[i-1]);以及大于划分元素的元素(放在 a[i],……,a[r])。于是,排序可以通过两个递归调用来完成,一个调用针对较小键,另一个调用针对较大键。为了达到此目的,程序将等于划分元素的键放在左边的 I 和 II 之间,右边的 rr 和 r 之间。在划分循环中,扫描指针停止,而且交换 i 和 j 处的项之后,她检查每个项是否等于划分元素。如果左边的一个元素等于划分元素,将它交换到数组的左部分;如果右边有一个元素等于划分元素,将它交换到数组的右边部分。具体函数代码如下:

函数:void sort(int a[],int l,int r)

形参说明:a 为接收待排序数组的首地址,l 为数组的起始下标,r 为数组的末下标。

```
void quicksort(int a[], int l, int r)
{ int i,j,k,p,q;
    int v,t; /*v 中存放标准键值,t 用于交换*/
    if(r<=l) return;
    v=a[r]; /*令最后一个元素为键值*/
    i=l-1; /*下面的语句是++i*/
    j=r; p=l-1; q=r; /*p,q 分别为大于和小于的两端位置*/
    for( ; ; )
    { while(a[++i]<v); /*在左端检测,如果遇到比 v 大的停止*/
        while(v<a[--j]) if(j==l)
```

```
            break; /*如果遇到比v小的停止*/
                if(i>=j) break;
                t=a[i]; a[i]=a[j];a[j]=t; /
            *交换检测到的两个值*/
                if(a[i]==v) { p++; t=a[p];
            a[p]=a[i]; a[i[=t; }
                if(v==a[j]) { q--; t=a[q];
            a[q]=a[j]; a[j]=t; }
        /*分别把与键值相等的移到两端*/
        }
        t=a[i];a[i]=a[r];a[r]=t;
        j=i-1; i=i+1;
        for(k=l; k<p; k++,j--) {t=a[k];a
    [k]=a[j];a[j]=t;}
        for(k=r-1;k>q;k--,i++) {t=a
    [k];a[k]=a[i];a[i]=t;}
            /*把等于键值的元素分别放到恰
当的位置*/
    quicksort(a, l, j); /*递归,对左子序列作划
分*/
    quicksort(a, i, r); /*递归,对右子序列作划
分*/
    }
```

本算法的实现要求在交换循环中只添加两个 if 语句,通过将等于划分元素的键放到位,它只需要两个 for 循环来完成划分。要维护这三个划分,与其他方法相比,似乎它需要的代码更少。更重要的是,它不仅以一种尽可能有效的方式来处理重复键,而且在无重复键时,带来的额外开销最少。

5 总结

总之,在进行快速排序的过程中,我们采用的是递归算法,在解决大问题化小问题的时候它显得尤为重要。同样,对于小问题我们可以采用低开销的直接插入型算法,这样通过混合算法提高了总的效率。在遇到特殊性问题的时候,我们同样采用具体问题具体分析的办法,通过不断的分析与改进,最终取得一个最佳的效果。

参考文献

1 徐孝凯、贺桂英,数据结构[M],北京:清华大学出版社,2004.

2 廖荣贵,数据结构与算法[M],北京:清华大学出版社,2004.

3 美 Robert Sedgewick. C 算法. 周良忠译注北京:人民邮电出版社 ,2004.

4 王刚,数据结构[M],北京:清华大学出版社,2004.