基于极值索引的数据排序算法

胡新帮* 汤志伟

(电子科技大学人文社科学院 成都 610054)

【摘要】提出一种新的数据排序算法，将数学极值的求解原理与数据排序结合，把极小值的概念扩展到记录的序列中，并按数据的排列规律，建立了极小记录索引，通过索引快速搜索待排序列中的记录，对待排序列快速的排序。该算法的最大时间复杂度$T(n)$为$O(n\log n)$和空间复杂度$O(n)$，在提高排序效率的同时，保证了排序结果中的相同大小记录之间相对位置的稳定。

关 键 词 排序； 极值索引； 时间复杂度； 空间复杂度

中图分类号 TP311.5； TP312 文献标识码 A

Data Sorting Algorithm Based on Extremum Index

Hu Xinbang Tang Zhiwei

(School of Humanities and Social Science, UEST of China Chengdu 610054)

Abstract This thesis proposes and discusses a new internal sorting algorithm, which combines mathematics extremum principle with data sorting algorithm. The algorithm expands the concept of minimum, establishes minimum extremum record index and can search very fast record from sequences through indexes of extremum record. As compared with the traditional internal sorting algorithms, the time complexity $T(n)$ of the algorithm is $O(n \log n)$ at the most and it's space complexity is $O(n)$. The algorithm guarantees the stability of data sorting while improves the efficiency of sorting.

Key words sorting; extremum index; time complexity; space complexity

排序是计算机程序设计中的一种重要操作，其功能是将一个数据元素(或记录)的任意序列重新排列成一个按关键字有序的序列。根据排序过程中涉及的存储器不同，可将排序方法分为内部排序和外部排序[1]。内部排序的方法很多，但就其全面性还没有最好的方法，每一种方法都有各自的优缺点，适合在不同的环境下使用，如果按照排序过程中依据的不同原则对内排序方法进行分类，则可分为插入排序、交换排序、选择排序、归并排序和技术排序等五类；如果按照排序过程中所需的工作量来区分，则可分为三类：1) 简单的排序方法，其时间复杂度为$O(n^2)$；2) 先进的排序方法，其时间复杂度为$O(n\log n)$；3) 基数排序，其时间复杂度为$O(dn)$。本文提出一种平均时间复杂度为$O(n)$的极值索引排序算法，该算法排序效率比时间复杂度为$O(n\log n)$的排序算法有了很大的提高。

1 极值索引排序原理

1.1 理论基础

根据数学极值原理和任意一组无序数据的组合队列都有局部区间(例如从第n个数据到第m个数据的区间表示为$[n,m]$)递增或递减或不变的特点[2]，本文将关于极小值的概念推广到非连续数值的数据记录排列中，

2003年5月5日收稿

* 胡新帮 男 21岁 学士 主要从事信息系统方面的研究；汤志伟 男 34岁 博士 副教授 主要从事信息系统和信息管理方面的研究

将处在递减区间向非递减区间过渡点的记录或处在非递减区间的首记录和处在递减区间的末记录定义为极小记录。即如果存在一个记录其关键字大于其前一条记录的关键字且小于或等于其后一条记录的关键字，或是首记录关键字不大于其后一个记录的关键字或末记录的关键字小于其前一个记录的关键字，该记录称为极小记录(记录的大小称谓与其关键字的大小统一)。由极值原理得出以下推论：1) 整个待排序列所有极小记录的集合必然包含最小记录；2) 最小记录所相邻两个记录中的较小记录和极小记录集合中的次小记录，其中较小的一个为整个待排序列中的次小记录。

1.2 极值索引排序原理

索引排序算法是基于上面的基本推论，充分利用了数据的排列规律，并结合数学中极值求解方法的应用，首先从一组记录中找出一组极小记录的集合，然后对这一集合进行排序，在选出极小记录集合并利用索引链存储各个记录位置的同时，对这些极小记录集合进行排序。

排序第一周期首先对待排数据进行遍历扫描，找出其所有极小记录的集合，对每个极小记录在队列中所处的位置标记索引，并对找出的极小记录重新组列排序，这是一个递归的过程，直到极小记录的集合序列呈现简单的递增或递减情况，或是只有一个递增区间和一个递减区间时，找出最小记录，以最小记录为索引对待排记录进行排序：找出最小记录相邻的一个较小记录a，将其和极小记录队列的下一个记录(次小记录)b比较，a与b两个记录中较小的为整个序列的次小记录，然后比较a与b中较大记录与其相邻记录，看其是否是极小记录，如果是则加到极小记录集合中。以上是排序的一次循环，找出次小记录后，将最小记录提前到待排序首位，下次循环就不再考虑最小记录，而以次小记录及其相邻记录排序的基准，按照上述过程进行排序，依此类推，直到排序完成，该算法的排序步骤如图1所示。

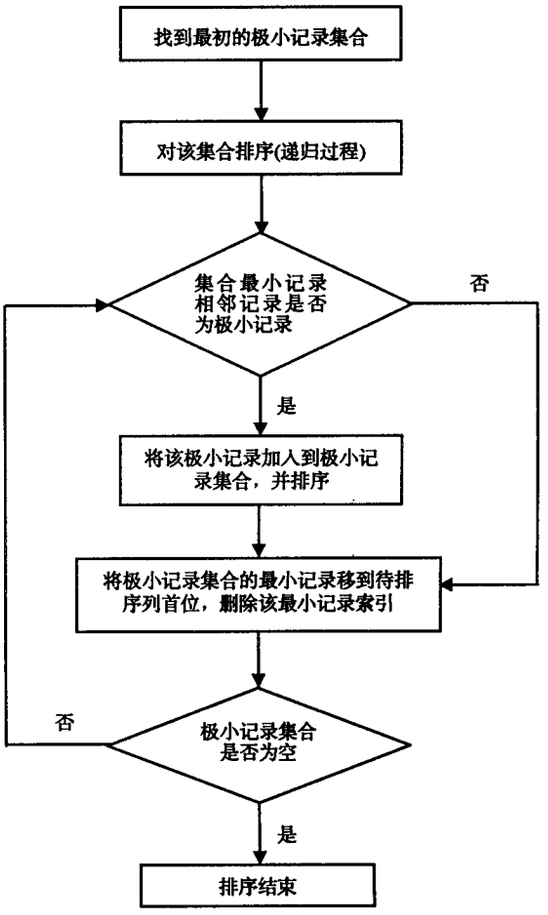

图 1　索引排序算法的排序步骤

2 算法伪代码与时间复杂度

2.1 基本数据结构定义

为了减少数据的移动量，并实现数据位置的快速转换，本算法使用双向链表记录，其基本结构体定义如下：

```
LinkStruct
    {                              //The key value of this structure or Data to sort,\
     Var          key;             //can be float or int or other datatype
     LinkStruct   *prior;          //Point to the prior structure
     LinkStruct   *next;           //Point to then next structure
    };
```

索引的结构也采用双向链表结构，同时使用一个指向待排数据结构体的指针用于记录其位置，索引结构体的定义如下：

```
IndexStruct
      {                             //**The Structure of Index
       LinkStruct     *Link;        // Point to Data Link Structure
       IndexStruct    *prior;       // Point to the Prior Index
```

```
    IndexStruct    *next;          // Point to the Next Index
  };
```

2.2 算法实现的伪代码

算法的实现分为两部分：1) **排序主函数**(MainSort)，**为实现排序的直接调用函数；**2) **索引序列排序**(IndexSort)，**采用递归调用对索引进行排序。由于两个函数的实现思路基本相同，所以只给出主函数的实现代码，采用类**c/c++**伪代码编写，其算法实现如下：**

```
MainSort(LinkStruct *SLink)
{   LinkStruct * p;
    LinkStruct * q;
    IndexStruct     Index;
    IndexStruct     * pIndex;
    p=&SLink;
    pIndex=&Index;
while(p->next!=null)
 {if   Minimum(p)   //loop for   Minimum Index
                {
                 if   (pIndex->Link!=null)
                   pIndex->next=new IndexStruct;
                 pIndex->next->prior=pIndex;
                 pIndex=pIndex->next;
                 pIndex.Link=p;
                };
    p=p->next;
}
if (Index->next->next)                  // (if) The index need be arranged in an order again or not /
                IndexSort(&Index);     // Sorting   the index
p=&SLink;
while (Index->Link->next!=null)
{
      p->next=Index->Link;
      p->next->prior=p;
      p=p->next;
      if (Index->Link->prior.key <= Index->Link->next.key)
                                Index->Link=Index->Link->prior;
                    else      Index->Link=Index->Link->next;
      if (Index->next->Link == Index->Link) || (Index.->Link.key > Index->Link->prior.key)||\
                               (Index.->Link.key < Index->Link->next.key)
                //(if) The   Current Index and the Next Index
                // point to the same LinkStruct or not Minimum    then delete one
                               {
                                Index->next=Index->next->next;
                                delete Index->next->prior;
                                Index->next->next->prior=&Index;
                               };
```

```
    pIndex=&Index;
    while(pIndex->Link.key >pIndex->next->Link.key)     // Sorting Index
    {
        pIndex->prior->next=pIndex->next;
        pIndex->next ->prior =pIndex->prior;
        pIndex->prior=pIndex->next;
        pIndex->next=pIndex->next->next;

    }
};
}
```

2.3 时间复杂度

在最好的情况下，只需对记录进行一组n次比较即可；在最坏的情况下，对n个记录组成的序列$\{R_1,R_2,R_3,\cdots,R_n\}$，最多有$n/2$(取整)个极小记录，栈的最大深度为$\log_2 n$，对序列的递归索引总次数应为$n+(n/2)+\cdots+1=2n$，当处于最坏情况时，假设一个记录$R_{2i}$为极小记录，则有$R_{2i}<R_{2i}+1$，$R_{2i}<R_{2i}-1$，当找出$R_{2i}$而将$R_{2i+1}$加入到极值序列时，只需进行一次的比较就可知道其不是极小记录而进入下一个步骤，随着索引记录的减少会出现一定次数的极小记录插入排序，但次数有限，为了减少程序的递归深度和索引的排序次数，当一次索引的索引记录数大于$n/4$时，进行2组的极小记录前移，将记录序列的极小记录集合分组前移，基本操作的次数也是$2n$。而使用一个已经升序排列的索引对一个待排序列进行排列，需进行n次比较和n次移动，在最坏的情况下，基本操作的执行次数为$4n$，所以时间复杂度$T(n)=O(n)$。

2.4 空间复杂度

在最坏的情况下，对n个记录组成的序列最多有$n/2$(取整)个极小记录，栈的最大深度为$\log_2 n$，则n个记录中最多包含$n/2$个极小记录，即最多为其建立$n/2$个索引节点，总共需要$(n/2)+(n/4)+\cdots+1=n$个索引节点，所以该算法的空间复杂度为$O(n)$。

3 极值索引排序过程分析

假如待排的一组记录的初始排列为：R_1 (49)，R_2 (38) ，R_3 (38)，R_4 (65)，R_5 (97)，R_6 (76)，R_7 (13)，R_8 (27)，R_9 (15)，R_{10} (49)，其中，对于$R_i(x)$(i=1,2,⋯,10)，i为记录的位置编号，x为关键字。下面对上述的待排记录队列进行排序：

1) 找出最小记录，对记录关键字进行比较得出R_2(38)、R_7(13)、R_9(15)为满足条件的极小记录，对这3个记录再次进行递归排序，极小记录集合的升序排列为R_7(13)、R_9(15)、R_2(38)，编号分别为1、2、3，其中R_7(13)为最小记录；

2) 找出次小记录，并判断其相邻记录是否为极小记录，首先比较R_7(13)的相邻记录R_6(76)和R_8(27)，得出R_8(27)较小，即在R_7(13)所处的区间[5，8] 中R_8(27)为次小记录，而在极小记录集合中的次小记录为R_9(15)，其次比较两个记录R_8(27)和R_9(15)，得出整个待排序列的次小记录为R_9(15)，同时将R_7(13)移动到队列的首位，最后比较R_8(27)与其相邻的两个记录R_6(76)(R_7(13)已经移到R_1 (49)的前面）和R_9(15)之间的关系，可判定R_8(27)非极小记录；

3) 重新组合索引，准备下一轮的排序，R_7(13)已经提前到队列的首位，对R_7(13)的索引的节点删除，只剩下对R_9(15)、R_2(38)的索引，对R_9(15)、R_2(38)的索引编号依次变为1、2，以R_9(15)为下次排序的基准，进入下一轮的排序。

同理类推，具体示意如表1所示，每两行为一组，第一行为记录的当前排列顺序，第二行的数字是对应极小记录的大小标号。

表1 待排记录队列进行排序过程分析表

次数	待排记录队列的当前排列顺序
1	R_1(49)，R_2(38)，R_3(38)，R_4(65)，R_5(97)，R_6(76)，R_7(13)，R_8(27)，R_9(15)，R_{10}(49) 3 1 2
2	R_7(13)，R_1(49)，R_2(38)，R_3(38)，R_4(65)，R_5(97)，R_6(76)，R_8(27)，R_9(15)，R_{10}(49) 2 1
3	R_7(13)，R_9(15)，R_1(49)，R_2(38)，R_3(38)，R_4(65)，R_5(97)，R_6(76)，R_8(27)，R_{10}(49) 2 1
4	R_7(13)，R_9(15)，R_8(27)，R_1(49)，R_2(38)，R_3(38)，R_4(65)，R_5(97)，R_6(76)，R_{10}(49) 2 1
5	R_7(13)，R_9(15)，R_8(27)，R_2(38)，R_1(49)，R_3(38)，R_4(65)，R_5(97)，R_6(76)，R_{10}(49) 2 1
6	R_7(13)，R_9(15)，R_8(27)，R_2(38)，R_3(38)，R_1(49)，R_4(65)，R_5(97)，R_6(76)，R_{10}(49) 1(值相等则位置靠前的优先) 2
7	R_7(13)，R_9(15)，R_8(27)，R_2(38)，R_3(38)，R_1(49)，R_4(65)，R_5(97)，R_6(76)，R_{10}(49) 2 1
8	R_7(13)，R_9(15)，R_8(27)，R_2(38)，R_3(38)，R_1(49)，R_{10}(49)，R_4(65)，R_5(97)，R_6(76) 1 2
9	R_7(13)，R_9(15)，R_8(27)，R_2(38)，R_3(38)，R_1(49)，R_{10}(49)，R_4(65)，R_5(97)，R_6(76) 1
10	R_7(13)，R_9(15)，R_8(27)，R_2(38)，R_3(38)，R_1(49)，R_{10}(49)，R_4(65)，R_6(76)，R_5(97) 1
11	R_7(13)，R_9(15)，R_8(27)，R_2(38)，R_3(38)，R_1(49)，R_{10}(49)，R_4(65)，R_6(76)，R_5(97)

从过程举例可以看出，极值索引排序算法的时间耗费主要在极小记录位置索引的建立，一旦极小记录位置索引建立并排序完成，最多进行两次关键字比较和一次记录移动就可以完成一个记录的排序。通过快速的定位记录位置，减少无效的记录移动次数，可以快速的进行排序。从排序过程中记录的排列和排序的结果可以看出，该算法较稳定。

4 结 束 语

综上所述，极值索引排序算法是一种效率很高的排序算法。极值索引排序的过程和堆排序很相似，两者的排序具备一定的相通性，对于极值索引排序算法在最坏的情况即数据呈现堆排列时，排序过程是堆排序过程的重现。但这两种排序算法是完成不同的排序，堆排序要求数据呈现堆排列，排序前需对数据进行预处理，无法根据数据的实际排列特点调整排序过程，而极值索引排序算法恰恰相反，在效率和通用性方面有突出的优点，且在不同的场合下其排序效率仍然存在提升空间。

本文研究工作得到校青年科技基金(L08011201 YF021003)资助，在此表示感谢。

参 考 文 献

[1] 严蔚敏, 吴伟名. 数据结构[M]. 北京: 清华大学出版社, 2002

[2] 同济大学数学教研室. 高等数学(第三版)[M]. 北京: 高等教育出版社, 1988

编 辑 徐培红