小 型 微 型 计 算 机 系 统
Journal of Chinese Computer Systems

2012 年 9 月 第 9 期
Vol. 33 No. 9 2012

一种自定义顺序的字符串排序算法

张海军[1,2],潘伟民[1],木妮娜[1],栾 静[1]

[1](新疆师范大学 计算机科学技术学院,乌鲁木齐 830054)
[2](中国科学技术大学 计算机科学与技术学院,合肥 230027)
E-mail: ustczhj@mail.ustc.edu.cn

摘 要: 现有的排序算法很难实现自定义顺序的字符串排序,提出一种自定义顺序的字符串快速排序方法. 在应用连续编号定义字符排序顺序的基础上,使用哈希表结构将字符串转换成对应的整型数组,以字符的最大编号作为基数排序算法的新基数,实现字符串的基数排序. 分析和实验表明,本文方法可有效实现自定义顺序的字符串排序,是一个时间和空间复杂度都是线性的排序算法,比快速排序(Quick Sort)具有更好的时间性能,且可以方便地推广到其它语言的字串排序中.

关 键 词: 字符串排序; 自定义顺序; 基数排序; 哈希表;
中图分类号: TP391 **文献标识码**: A **文 章 编 号**: 1000-1220(2012)09-1968-04

A String Sort Algorithm in Custom Character Order

ZHANG Hai-jun[1,2], PAN Wei-min[1], MU Ni-na[1], LUAN Jing[1]

[1](*School of Computer Science and Technology, Xinjiang Normal University, Urumqi* 830054, *China*)
[2](*School of Computer Science and Technology, University of Science and Technology of China, Hefei* 230027, *China*)

Abstract: Existing sort algorithms are difficult to implement string sort in a custom order of characters. This paper presents a fast string sort method in custom character order. On the basis of the consecutive numbers which used to define the custom order of characters, the hash table structure is employed to convert each string into corresponding array of integers. By taking the maximum number of characters as the new radix, the radix sort algorithm is used to implement fast sort of strings in custom order. Theory analysis and experiments show that the method of this paper can easily achieve string sort in custom order in linear time and space complexity. This method has a better time performance than that of Quick Sort algorithm, and it can easily extend to string sort applications of other languages.

Key words: string sort; custom order; radix sort; Hash table

1 引 言

排序是数据处理领域的一种基本操作,其主要作用是加快检索和归并速度. 字符串排序是自然语言处理中的重要基础技术,在基于大规模网络语料的网络新词识别、有意义串检测、舆情监测等领域,字符串排序算法都有重要应用; 在大规模字符串的外部排序过程中,字符串的内部排序算法更是起到了至关重要的作用.

数据排序算法可以被分成两大类: 基于比较和基于非比较的排序算法. 前者涉及的算法较多,主要有冒泡排序算法(Bubble Sort)[1]、选择排序算法(Selection sort)、插入排序算法(Insertion Sort)、快速排序(Quick Sort)[2]以及归并排序(Merge Sort)等,前 3 种方法的时间复杂度(Time Complexity)均为 $O(n^2)$,空间复杂度(Space Complexity)为 $O(n)$; Quick Sort 是目前最经典的排序算法,其时间复杂度在数据均匀情况下是 $O(n\lg n)$,最坏情况下蜕变为 $O(n^2)$,空间复杂度为 $O(n)$; 归并排序算法的平均时间复杂度也为 $O(n\lg n)$,但同快速排序算法相比,归并排序在时间复杂度上具有更大的常量系数,其空间复杂度为 $O(2n)$; 桶排序[3,4]方法是一种特殊的排序方法,非常适合处理分布均匀的数据,是时间复杂度为 $O(n)$ 算法,但对极不均匀数据则退化成复杂度为 $O(n^2)$ 的排序算法. 何文明等[5]提出了一种针对复杂数据的分档排序方法,可有效提升特定数据的排序效率,但对于汉字串排序,性能还有改进的空间. 基数排序[6](Radix Sort)是典型的基于非比较的排序算法,其时间复杂度为 $O(dn)$,空间复杂度为 $O(2n)$,虽具有较好的性能,但通常应用于数值排序领域,在一定程度上限制了其使用范围. 王向阳[7]将基数排序与映射链接思想相结合,提出了基数分配链接的排序方法,该算法的时间复杂度为 $O(n)$,排序效果与数据分布无关.

上述算法所进行的排序都是基于待排序对象的机内编码顺序,产生的排序结果无非有正序和逆序两种,如 4 个字符: "A"、"B"、"C"、"D"通过排序只能产生"A B C D"或"D C B A"两种特定的序列,无法产生按照用户自定义顺序进行排序的结果,如: "D B A C"等排序序列就无法通过前述排序方法

收稿日期: 2011-04-22 基金项目: 国家自然科学基金项目(61163045,31040050)资助; 新疆师范大学博士后科研启动基金项目(XJNUBS1111)资助. 作者简介: **张海军**,男,1973 年生,博士,研究方向为自然语言处理、新词识别技术; **潘伟民**,男,1963 年生,博士,教授,研究方向为知识库构建、网络系统; **木妮娜**,女,1963 年生,博士,副教授,研究方向为模式识别、人工智能; **栾 静**,女,1962 年生,博士,教授,研究方向为自然语言处理、舆情监测.

有效产生,对于由多个字符所构成字串的自定义顺序排序更是无从谈起. 毕竟在某些应用情况下,需要根据用户定义的顺序进行数据排序,如在检索时可能需要以特定顺序进行结果输出,但目前的排序方法都难以满足这个要求. 为此,本文在深入研究排序算法和相关数据结构的基础上,以中文字符串排序为例,提出了一种用户自定义顺序的高效字符串排序方法,并基于理论和实验对该算法进行了分析和讨论. 该算法的重要特点是,可实现用户自定义顺序字串排序,且时间和空间复杂度都是线性的.

2　一种自定义顺序的字符串排序算法

自定义顺序字符串排序是指根据用户需求,重新设定字母表中字符的大小顺序,并实现按照新顺序所进行的排序. 一般地,假设有 n 个字符串序列$\{S_1,S_2,\cdots,S_n\}$,每个字串 S_i 中都有 d 个字符组成$(C_i^0,C_i^1,\cdots,C_i^{d-1})$,则字符串的自定义顺序有序定义为,对序列中的任意两个字符串 S_i 和 $S_j(1\leqslant i<j\leqslant n)$ 都满足下列有序关系:$(C_i^0,C_i^1,\cdots,C_i^{d-1})\leqslant(C_j^0,C_j^1,\cdots,C_j^{d-1})$,且其中 $C_i^0\leqslant C_j^0,C_i^1\leqslant C_j^1,\cdots,C_i^{d-1}\leqslant C_j^{d-1}$ 是用户自定义的字符顺序,而非原有的机器内码字符排列顺序. 通过自定义顺序排序可以实现按照用户特定要求实现任意确定顺序的字符串排序.

若以基于数值比较的排序方法作为研究基础,每定义一种排序顺序时都需要对有关算法进行复杂的改写,难以实现灵活的自定义字符顺序,且字符串排序效率也有待改进. 鉴于上述问题,本文考虑以不需要数值比较的基数排序(Radix Sort)方法作为算法基础,并结合有效的数据结构以实现用户自定义顺序字串高效排序.

2.1　Radix Sort 算法描述

基数排序是基于非比较排序的经典算法,主要思想是通过关键字来实施排序对象位置的映射. 针对多个关键字$(K_i^0,K_i^1,\cdots,K_i^{d-1})$,在排序时需要进行多轮的待排对象位置映射,每一轮使用其中的一个关键字来映射对象位置. 在实施基于多关键字的对象位置映射时,目前有两种具体的排序策略. 一种是从最高位关键字 K^0 开始第一轮位置映射,然后依次采用 $K^1,K^2,\cdots$,直至 K^{d-1} 进行排序对象的位置映射,当映射结束后,也即完成了基数排序,这种被称为最高位优先策略;另一种是从最低位关键字 K^{d-1} 开始进行第一轮对象位置映射,然后依次采用 $K^{d-2},K^{d-3},\cdots$,最后采用 K^0 进行排序对象的位置映射,这种被称为最低位优先策略[8].

最高位优先策略更接近于人们的思维习惯,但在进行位置映射时若某位关键字相同,需要参阅更低位的一个或多个关键字方可有效实施基数排序;而采用最低位优策略,在排序对象位置映射时只需考虑当前的关键字,无需参阅其它位置,即可高效实施对象排序. 但应用最低位优先策略的前提是基于单关键字进行对象位置映射的算法必须是稳定的. 基数排序算法(Radix Sort)的具体实现描述为[6]:

Radix-Sort(***A***, ***d***)

FOR i: =1 **to** d

DO 依据第 i 位关键字对数组 A 中的排序对象进行位置映射

为了获得更好的排序效率,基于关键字进行对象位置映射的方法选用计数排序算法(Count Sort),以最大限度地降低基数排序时间,此时 Radix Sort 的时间复杂度为 $O(d(n+K))$,式中 n 为待排序对象的数量,d 为待排对象关键字的位数,K 表示排序中数值的基数. 若 $n>>K$,则基数排序所需时间可表示为 $O(dn)$.

2.2　基于非比较方法的中文字串排序

基数排序使用计数排序算法来实现每一位关键字的排序,但计数排序的处理对象是整数序列,而汉字串却是汉字序列,因此无法直接应用计数排序算法对汉字串进行位置映射. 若能在排序前将汉字序列转化为整数序列,即可实现字符串的基数排序. 为了不影响整体排序效率,汉字序列与整数序列之间要有足够快的变换速度,且需实现二者之间的一一映射.

根据以上要求,本文提出使用哈希表结构来实现中文字符串与整型数字序列之间的快速映射. 哈希表检索的基本思想是以待检索的关键字为自变量,通过哈希函数计算出函数值,并把该值解释为一块连续存储空间中的确定单元地址,并将该记录存储到这个单元中. 将不同的关键字映射到不同的存储空间并尽可能保持不发生存储冲突,是哈希表检索结构设计的关键和重点,除选择合适的哈希函数来解决以外,使用合理的冲突处理方法也是减少冲突提高检索效率的有效手段. 哈希表检索最显著的优点是效率高,其时间复杂度仅为 $O(1)$,与数据量无关;但这样高的检索效率是牺牲了存储空间来取得的,哈希表结构需要约 2 倍的存储空间.

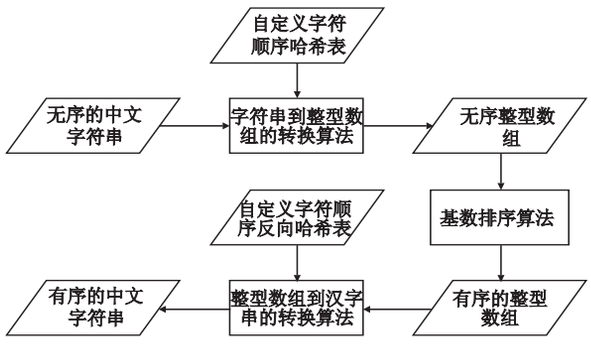

图 1　自定义顺序字符串排序算法流程图

Fig. 1　Flow chart of custom order sort algorithm

根据哈希表检索的特点,可按照自定义顺序对汉字进行整数编号来构造字符顺序字典,利用编号大小来表示字符顺序(编号小者代表字符值小,编号大者代表字符值大)如此就实现了对汉字排序顺序的重新定义;然后将汉字及其对应的编号装入哈希表中以提高字串到编号序列的转化速度. 为满足基数排序要求,通过哈希表检索方式,将字符串转化为同等长度的整型数组,数组中的元素用于存储所对应位置的汉字编号,因汉字与编号一一对应,所以转化后的字符串与整型数组是一一对应的,每个字符串都由一个特定的整型数组代表. 将整型数组看成是位数固定的数字(位数为整型数组的长

度,数组中的每个元素都相当于整数中的一位),然后使用基数排序方法对代表字符串的整型数组进行排序. 这个过程实际上是对普通的基数排序中所用的"基数"进行了改变,如整数排序的基数是10,而本文整型数组的基数变成了待排字串字符表中字符总数(如英语为26,GB2312-80汉字集合为6763等). 在完成了代表字符串的整型数组排序后,再经过反向查表将所有的整型数组转化为对应的字符串序列,至此也即完成了自定义顺序字符串的排序过程. 上述算法的具体流程参见上页图1所示.

本算法提出应用整型数组来表示汉字串,对串中的字符,从哈希表(Hash Table)中检索其编号并装入数组的对应位置,这样做的目的是可直接在基数排序的基础上实现自定义顺序的字符串排序. 下面列举一些字符串与整型数组的转换示例,参见表1所示.

表1 字符串和数字序列对照表[1]

Table 1 Table between strings and number sequences

字符串示例	整型数组内容
阿贾克斯	[2][2093][2558][4570]
男单冠军	[3453][829][1580][2474]
中国财经	[6422][1643][384][2314]
中国财税	[6422][1643][384][4545]
注册公司	[6494][418][1486][4565]
注明出处	[6494][3363][635][655]
电脑	[945][3475][0][0]
自己想	[6593][2028][5313][0]

表1中列举了一些示例字符串和与之对应的整型数组,在实施基于整型数组的基数排序时,将每个整型数组当作一个整数,数组中的每个元素当作整数中的一位来处理即可. 汉字串串长不同会导致对应的数组长度不一致,为便于实施基数排序,本文应用等长数组(长度为最大串长)来表示汉字串,并采用左对齐方式,与短串相对应数组的右侧空缺元素用0补齐.

3 自定义顺序字符串排序算法的性能分析

首先分析算法的时间复杂度(Time Complexity),设n为待处理字符串的条数,d为最大串长;哈希表(Hash Table)的查表时间复杂度为 $O(1)$,将字符串转换为对应的整型数组的时间复杂度是 $n \times d \times O(1) = O(dn)$,基数排序部分的时间复杂度是 $O(dn)$,将整型数组序列转换为字符串序列的时间复杂度是 $n \times d \times O(1) = O(dn)$,因此本文自定义顺序排序算法的时间复杂度为 $O(dn) + O(dn) + O(dn) = O(dn)$. 虽然同普通的基数排序方法具有相同的时间复杂度,但自定义顺序字符串排序方法因需要两步附加转换,复杂度的常量系数约是普通基数排序的3倍.

对于算法的空间复杂度(Space Complexity),由于采用简单连续的字符编码方法,整型数组同字符串所占的空间是相同的. 对于汉字采用GB2312-80标准编码,需要2Byte内存;而汉字编号采用占用空间为2Byte的整数来表示,可见汉字串与代表它的整型数组所使用的内存相同;对于英文字符使用ASCII编码,每个字符占用一个字节,其编号可用一个字节来表示,这样字符串所占空间与整型数组仍然相同. 因此,根据基数排序算法的相关理论,本文排序算法的空间复杂度为 $O(2n+M) + O(M)$,其中 M 为字符的最大编码值(也就是待处理字符集中的字符总数,对于汉字的GB2312编码 M 值不超过7000,对于英语 M 值不超过26),该部分内存用于计数排序的辅助存储,式中 $O(M)$ 表示存储字符编号的哈希表结构所占的内存. 若存在 $n >> M$ 时,该算法的空间复杂度就收敛为 $O(2n)$.

依据上述分析,自定义顺序字符串排序以基数排序方法作为基础,不需要数值比较,其时间复杂度是 $O(dn)$ 空间复杂度是 $O(2n)$,是线性时间和空间的排序方法. 从理论上讲,应该比包括快速排序算法(Quick Sort)在内的基于比较的排序方法具有更高效率.

4 实验和讨论

4.1 实验条件

根据第2节中的算法描述,使用Microsoft C# 2005进行了算法实现. 实验的硬件环境是CPU主频2.6GHz,内存为2GByte,操作系统采用Windows XP SP3,所用实验数据来源于搜狗实验室(Sogou Labs)提供的大规模网络语料[9],在其中抽取了1600万条中文字符串(字串最大长度为3)作为实验分析之用. 在实验时,根据自定义顺序对6763个GB2312汉字进行编号,并将字符编号保存在文本文件中,以便于将之装入哈希表中实现字符串和对应的整型数组之间的快速转换. 为更好地分析算法性能,采用递增规模的语料进行了多组平行实验,具体数据参见表2.

表2 自定义顺序排序方法对比实验数据

Table 2 Comparative data of custom order sort method

数据规模	排序时间(ms)				
(万条)	A	B	C	D	E
100	1266	1266	1266	1266	1266
200	2703	2641	2640	2641	2594
400	5406	5422	5438	5422	5391
800	11938	12015	11406	11406	11234
1100	15985	16062	16031	15938	17063
1600	24922	24031	23922	23641	23797

4.2 数据分析

通过查看排序结果,发现该算法是严格地按照自定义的顺序实现字符串排序的. 为减少实验中偶然性影响,表2对相同数量字符串进行了平行排序试验,并且基于递增规模的字符串实施了性能对比. 从横行上看,虽存在偶然因素,但同一

[1] 表中字符的顺序是其机内码顺序. 当然用户可以自行定义不冲突的字符编号顺序.

规模数据的排序时间差别不大; 从纵向上比较,排序时间随着排序工作量的递增而增长,且随着工作量的加倍,排序时间也在加倍. 为进一步分析排序时间和数据规模之间的关系,绘制了二者的关系图,见图 2.

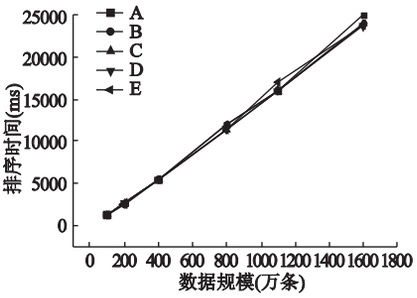

图 2　排序时间与数据规模关系图
Fig. 2　Diagram between sort time and data size

根据图 2 可知,本文自定义顺序排序算法的排序时间与字符串数量之间呈现线性关系,进一步验证了第 3 节中的分析结论,说明该算法是一个线性时间的排序算法.

为了进一步分析算法的内存使用情况,对递增语料的排序占用空间进行统计,结果参见表 3 所示.

表 3　算法的内存使用实验数据
Table 3　Experimental data of memory usage for improved algorithm

数据规模(万条)	内存使用(KB)	内存用量倍比
100	54672	
200	106144	1.941
400	209044	1.969
800	412428	1.973
1600	819268	1.986

表注: 内存用量倍比是指递增规模数据排序内存用量之比,即用后面的内存用量比前面的内存用量

从内存用量上来看,排序中内存的占用同语料规模之间在逐渐接近线性关系; 内存用量的倍比随着语料规模的增长在增加,逐渐向 2 逼近. 没有到达 2 的主要原因是在本文的改进算法中,存储字符编号的哈希表和基数排序中的辅助存储都要占用一定的内存,其数量级约为 $O(M)$,随着语料规模的增长,该部分内存在所占的比重在逐渐降低,所以内存用量倍比在向 2 靠近. 由此可见,本文方法在排序大规模数据时空间效率会更高.

4.3　算法横向性能比较

在目前用于字符串排序的算法中,快速排序(Quick Sort)比较典型,具有最快的排序速度. 针对相同语料,对本文算法同快速排序算法进行效率比较. 实验时本文算法的自定义顺序采用汉字字符的机内码顺序,这样可以确保同快速排序算法的排序结果一致. 实验数据参见表 4 所示.

表 4　不同排序方法的排序效率对比实验数据
Table 4　Comparative data of sort efficiency for different sort methods

数据条数(万条)	400	800	1600
本文算法时间(ms)	5422	11406	23797
快速排序时间(ms)	17266	36797	76390

根据实验结果,对递增规模的待排序字符串,自定义顺序排序算法比快速排序拥有更高效率; 且随着待排序字符串数量的增加,上述两种算法的排序效率差距在逐渐加大. 表 4 中数据很好的验证了第 3 节的理论分析,本文自定义排序是线性时间的排序算法,复杂度低于快速排序的 $O(nlgn)$,所以具有更快的排序速度.

从空间占用来看,本文算法的内存占用量大约为快速排序算法的 1.8 倍左右,可见其之所以速度较快,是由于使用了空间来换取时间的结果.

5　结论和进一步工作

本文通过连续数字编号来定义字符的排序顺序,使用哈希表结构来实现自定义顺序基础上的字符串和对应整型数组之间的快速转换,在基数排序的基础上实现字符串的非比较方法快速排序. 理论分析和实验结果都表明,本文自定义顺序排序方法可以实现对中文字符串的线性时间排序. 虽然本文方法以中文字符串排序为例,但其基本思想是通用的,可方便地将本文排序方法扩展到其它语种,实现字串自定义顺序高效排序.

本文在实现过程中,对不同串长的中文字符串使用相同长度的数组进行字串编号,在处理短串时会造成存储空间浪费,所以该算法更适合于串长相同或均匀长度的字符串排序. 后续研究在减少内存消耗的基础上,将该算法应用到大规模语料的外部排序中,以提高外部排序速度,改进新词识别中重复模式的提取效率.

致谢:

本文实验中采用了搜狗实验室的互联网语料库(SogouT),在语料获取过程中得到了倪剑莉老师的热心帮助,在此一并表示诚挚的谢意.

References:

[1] Owen A. Bubble sort: an archaeological algorithmic analysis1 [A]. In: Proc of the 34th SIGCSE Technical Symp on Computer Science Education, New York [C]. ACM Press, 2003: 1-5.
[2] Hore C. Quicksort [J]. The Computer Journal, 1962, 5(1): 10-16.
[3] Yang Lei, Huang Hui, Song Tao. The sample separator based distributing scheme of the external bucket sort algorithm [J]. Journal of Software, 2005, 16(5): 643-651.
[4] Yang Lei, Song Tao. The array based bucket sort algorithm [J]. Journal of Computer Research and Development, 2007, 44(2): 341-347.
[5] He Wen-ming, Cui Jun-zhi. New high efficient sorting algorithm by grading to character string data [J]. Mini-Micro Systems, 2004, 25(4): 698-701.
[6] Cormen T H, Leiserson C E, Rivest R L, et al. Introduction to algorithms (2nd Ed) [M]. Cambridge MA: MIT Press, 2001.
[7] Wang Xiang-yang. A new sorting method by base distribution and linking [J]. Chinese Journal of Computers, 2002, 23(7): 774-777.
[8] Yan Wei-min, Wu Wei-min. Data structure(C language version) [M]. Beijing: Tsinghua University Press, 1997.
[9] Sogou Lab. The internet corpora(SogouT) [EB/OL]. http://www.sogou.com/labs/dl/t.html, 2009-07-25.

附中文参考文献:

[3] 杨　磊, 黄　辉, 宋　涛. 桶外排序算法的抽样分点分发策略 [J]. 软件学报, 2005, 16(5): 643-651.
[4] 杨　磊, 宋　涛. 基于数组的桶排序算法 [J]. 计算机研究与发展, 2007, 44(2): 341-347.
[5] 何文明, 崔俊芝. 针对字符串等复杂数据的一种新的高效分档混合排序算法 [J]. 小型微型计算机系统, 2004, 25(4): 698-701.
[7] 王向阳. 任意分布数据的基数分配链接排序算法 [J]. 计算机学报, 2000, 23(7): 774-777.
[8] 严蔚敏, 吴伟民. 数据结构(C 语言版) [M]. 北京: 清华大学出版社, 1997.
[9] 搜狗实验室. 互联网语料库(SogouT) [EB/OL]. http://www.sogou.com/labs/dl/t.html, 2009-07-25.