第34卷 第10期 计 算 机 工 程 2008年5月
Vol.34 No.10 Computer Engineering May 2008
·软件技术与数据库· 文章编号：1000—3428(2008)10—0083—03 文献标识码：A 中图分类号：TP391.3

基于压缩后缀数组技术的搜索引擎

姚全珠，张 楠，杨增辉，田 元
(西安理工大学计算机学院，西安 710048)

摘 要：目前，搜索引擎的核心模块(索引器)均采用倒排文件结构，对短语查询的准确率较低。该文引入后缀数组技术进行全文索引，为克服全文索引时占用空间大的缺点，研究了压缩后缀数组技术，把后缀数组索引的大小压缩到了$O(n)$位，并给出应用压缩后缀数组索引的步骤和核心操作伪代码。对比实验表明，基于压缩后缀数组的索引比传统倒排文件索引的短语查准率提高了近20%。

关键词：压缩后缀数组；倒排文件；后缀数组；搜索引擎

Search Engine Technology Based on Compressed Suffix Array

YAO Quan-zhu, ZHANG Nan, YANG Zeng-hui, TIAN Yuan
(School of Computer Science, Xi'an University of Technology, Xi'an 710048)

【Abstract】 The core module of search engines, namely indexer, is usually based on inverted file. But this solution to solve phrase-search is in difficulty(the lower hitting rate). In this paper the Suffix Array(SA) are employed for full-text indexing. In order to overcome the disadvantage of large memory cost as with full-text indexing, research is done for Compressed Suffix Array(CSA). The paper presents the step of using CSA index and the false code of core operate. The experiments show that this technique, compared with inverted file, improves the hitting rate for phrase by 20%.

【Key words】 Compressed Suffix Array(CSA); inverted file; Suffix Array(SA); search engine

1 概述

目前，搜索引擎已经成为继电子邮件之后的互联网第二大应用。索引器是搜索引擎的核心模块，它的实现机制是区分搜索引擎层次的评判标准，其改进是提高查准率的唯一途径。不同的搜索引擎，处理索引的方法不同。目前，有3种建立索引的技术：

(1)签名文档：现在基本被淘汰。

(2)倒排文件：目前在主流搜索引擎中广泛使用。

(3)后缀树/后缀数组：正处于研究发展阶段。

倒排文件是一种面向单词的索引机制[1]。它的结构由两种元素组成：词汇表(vocabulary)和事件表(occurrence)。词汇表是文本中所包含的所有不同单词的集合。对于词汇表中的每一个单词，在文本中出现的所有位置都存储在一个列表中，所有列表的集合就称为事件表。

通常，词汇表的空间需求比较少，适合于放置在内存中。但事件表所占用的空间相对较大。由于在事件表结构中将涉及出现在文本中的每一个单词，因此所占用的空间是$O(n)$。事实上，即使去除停用词，也需要原始文本大小的30%~40%左右[2]。

对于单个词汇的查询来说，只要从词汇表中找到对应的单词就可以找到指向该单词的出现情况列表。其查询的时间复杂度为$O(\log n)$(n为词汇表的长度)，并且该词汇的出现情况列表可以直接作为搜索结果返回给用户。

倒排文件索引的优势是易于构造实现，在大小为n个字符的文件上进行索引可以在$O(n)$时间内构造完成，并且在关键词的搜索上具有良好的性能。但其缺点在目前的使用中也愈来愈明显，最主要的是不方便进行短语查询。另外，倒排索引是一种词索引，非常适合于英语文本，并不适合于中文、日语等。因为它们很难被划分成一个一个的词，中文分词就是一个目前还没有很好解决的问题。对于这些情况使用全文索引，即后缀数组[3]和后缀树[4]技术，可以检索到文本中的任何一个子字符串，不仅避开了中文分词，而且能高效地实现短语查询。

由于全文索引占用的空间要比词索引大得多，因此后缀数组技术还不能直接应用于搜索引擎中，必须对它加以改进。压缩后缀数组(compressed suffix array)技术的引入可以解决以上问题。

2 后缀数组和后缀排序

后缀是指从原字符串S中相应位出发直到字符串S结束的子字符串，如：原始字符串$S=abcd$，那么后缀$S_0=abcd$，$S_1=bcd$，$S_2=cd$，$S_3=d$。而后缀排序就是对一个原始字符串S的所有后缀按字典序排序。在上例中，对原字符串S做后缀排序(即对S_0, S_1, S_2, S_3进行排序)的结果是：S_0, S_1, S_2, S_3。

设一个文档(字符串T)长度为N。T的后缀数组(suffix array) s是指将T所有的后缀$Ts(0)$,$Ts(1)$, …, $Ts(N-i)$按照字符顺序(lexicographic order)排序而构成的数组。$S[i](0\leq i<N)$表示一个后缀$Ts(k)(0\leq k<N)$，而且$Ts(K_{i-1})\leq Ts(K_i)(1\leq i<N)$。

在构造索引文件时，要考虑检索时如何更快捷地找出所需的索引词条，这就涉及到后缀排序问题。在实际应用中原始字符串S(大文本)是非常巨大的，所以一般的快速排序算法不能满足需要，本文给出一种基于后缀数组的快速排序算法

基金项目：陕西省自然科学基金资助项目(2005F07)
作者简介：姚全珠(1960－)，男，博士、教授，主研方向：数据库与系统集成，网络技术；张 楠、杨增辉、田 元，硕士
收稿日期：2007-05-25 **E-mail：**qzyao@xaut.edu.cn

并以实验表明了这种方法的优越性。先给出几个必要的定义：

定义 1 索引数组 I(index arrays)是描述后缀数组位置的数组。如果数组 $I[0..n]$是后缀数组 Si 的索引数组，那么在 I 中定义了后缀数组 Si 在排序过程中的位置变化。其值是通过对后缀字符串中每个字符依次按照字典序比较得出的。

定义 2 信号数组V(signal arrays)是排序进展程度的标志。V是一个与S长度相等的数组，通过该数组中元素值是否相等来表示排序的结束与否。经过k次迭代后，如果V中存在相等的元素，则将具有相同V值的元素划分为一组，同时也表明这些元素的前k个字符一定是相同的；如果在V中，所有元素均不相等，则表示字符串已经有序，排序过程结束[4-5]。该数组必须满足以下 3 个条件：

(1)若 $Xi<Xj$，则 $V[i]\leq V[j]$；

(2)若 $Xi>Xj$，则 $V[i]\geq V[j]$；

(3)若 $Xi!=Xj$，则 $V[i]!=V[j]$。

定义 3 规模数组 L (size arrays)表示分组元素数量的数组，是为了提高排序算法的时间性能而引入的。对于无序的组 $L[f]=g-f+1$；对于有序的组：$L[f]=-(g-f+1)$。

在进行第 j 次排序时，$k=2j-1$，表示用小于 k 的步长来遍历所有的分组，第一组以索引 0 开始，元素个数表示为 $L[0]$，用 $V[s+k]$对组 $I[0..size-1]$中的后缀进行排序；下一组从索引 $g=size$ 处开始，对组 $I[g..g+S[g]-1]$中的后缀进行排序；当遍历了所有不同的分组后，将其划分为小于 $2k$ 有序，如果同一分组 g 中连续的 $I[i]$和 $I[i+1]$具有不同的值，即 $V[I[i]+k]!=V[I[i+1]+k]$，则在索引 i 和 $i+1$ 之间划分分组，同时更新每一分组中元素的个数，最后，更新 $V[I[i]]$的值。

算法的具体实现步骤如下：

(1)通过第一个字符对 Si 进行排序(按字典序)，结果保存在索引数组中，$j=1$, $k=2j-1=1$；

(2)在信号数组 V 中设置组号；

(3)对每一个未排序组和排序组，在规模数组 L 中设置组的长度(没有排好序的组用正数表示，排好序的组用负数表示，并且根据情况将连续的有序分组合并为一个分组)；

(4)用 ternary-split Quicksort(三分快速排序算法)处理每一个未排序数组，使用 $V[I[i]+k]]$更新后缀 i 的关键字；

(5)将 k 加倍，对未排序组中不等于关键字的部分递归调用该排序算法；

(6)如果L中的元素值的绝对值等于所有元素个数n，则说明所有元素已经有序，排序过程结束；否则，$j=j+1$,$k=2^{j-1}=2\times k$，转向(4)[4]。

实验：用一般的快速排序算法和本算法对 5 个不同大小的字符串文件进行排序，其排序所需的时间的对比结果如表 1 所示。

表 1　快速排序算法和基于后缀数组的快速排序算法比较

files 大小/B	排序时间/s	
	快速排序	基于后缀数组的快速排序
285	1	1
1 732	4	3
492 644	7 933	35
884 579	24 050	75
1 822 577	123 373	180

结果分析：当排序文本文件较小时，这两种算法所需的时间几乎相等，但是随着排序文件逐渐变大，该排序算法所需的时间越来越少于快速排序算法，即文件越大，其优势越明显。对于体积为 n 的文本文件，该算法的空间复杂度是 $O(5n)$，时间复杂度为 $O(n\log n)$。

分析原因，其优势来源于以下 3 个方面：

(1)一般的快速排序算法通过对比将待排序的元素分为两组(大于和小于)，而该算法将等于作为一种新的情况来看待，将待排序的元素分为 3 组(大于，等于，小于)，由于采用了分而治之的思想，减少了排序中递归调用的次数，降低了算法的时间复杂度。

(2)通过在递归排序过程中使 k 值的增加 1 倍，充分利用了前一次排序结果，并且将复杂的字符串排序问题转化为简单的数值比较。

(3)规模数组中元素值的正负来表示该组元素是否已经有序，如果有序，则可以越过该组，从而使排序进行的更快，过程更简单。

3　压缩后缀数组及其在搜索引擎中的应用

采用全文索引是由于后缀数组技术占用的计算时间并不大，但占用的空间比较大。如何减少全文索引占用的空间就成了应用后缀数组的重点。

压缩后缀数组技术，可以把后缀数组的大小压缩到$O(n)$位，克服了占用空间大的缺点。压缩后缀数组技术的一般步骤是[5](其中$n_0=n$，$SA_0=SA$)：

(1)生成一个有n_k位的位数组B_k，如果$SA_k[i]$是偶数，则$B_k[i]=1$；反之，如果$SA_k[i]$是奇数，则$B_k[i]=0$。

(2)定义一个函数Ψ_k，如果当且仅当$SA_k[i]$是奇数并且$SA_k[j]=SA_k[i]+1$时，$\Psi_k(i)=j$。如果$\Psi_k(i)=j$，就可以得出$B_k[i]=0$和$B_k[j]=1$。当$SA_k[i]$是偶数时，$\Psi_k(i)=i$。所以Ψ_k函数的取值可以用如下的公式表示：

$$\Psi_k=\begin{cases} j & \text{如果}SA_k[i]\text{是奇数并且}SA_k[j]=SA_k[i]+1 \\ i & \text{否则} \end{cases} \quad (1)$$

(3)定义函数$rank_k$，用它来计算B_k中 1 的数目。$rank_k(j)$等于B_k中前j位中的 1 的数目。

(4)将SA_k中所有的偶数都除以 2，结果组成一个新的关于$\{1,2,\cdots,n_{k+1}\}$的排列的数组SA_{k+1}，其中$n_{k+1}=n_k/2=n/2^{k+1}$。

通过以上 4 个步骤，后缀数组SA的长度就可以减半。经过$L=\mathrm{lb}(\mathrm{lb}n)$步后，后缀数组$SA$的长度就减少为原来的 $1/2^L$。通过式(2)，就可以从SA_L得到原来的后缀数组SA。

$$SA_k[i]=2\cdot SA_{k+1}[rank_k(\Psi_k(i))]+(B_k[i]-1) \quad (2)$$

这是一个递归调用的过程。在整个压缩过程中，在第k级($0\leq k<L$)，B_k，Ψ_k和$rank_k$都需要保存下来，但SA_k则不需要保存。在最后一级，即$k=L$时，SA_L就保存下来，而不再需要B_L,Ψ_L和$rank_L$。

从式(2)中可以看出从SA_L检索SA的过程是一个递归过程，所以检索函数lookup也是一个递归过程。用伪代码表示如下：

```
procedure lookup(i,k)
    if k=L then
        return SA_L[i]
    else
        return
2*lookup(rank_k(Ψ_k(i)),k+1)+(B_k[i]-1)
```

现分析 lookup 函数的时间复杂度如下：

lookup函数调用$j-i+1$ 次，如果$SA[i]$和$SA[j]$有 $\Omega((\mathrm{lb}n)^2)$个相同的前缀，那么总共的时间复杂度是$O(j-i+(\log n)^2\log(\log n))$。否则，总时间复杂度为$O(j-i+n^a)$，其中，$a$是常量，且 $0<a<1$，这时$SA[i]$和$SA[j]$有 $\Omega(\mathrm{lb}n)$ 个相同的前缀。

以上就是压缩后缀数组技术的理论依据。为了使该技术

能在搜索引擎中更好的应用，本文在此基础上提出 3 个新的操作。

(1)*inverse*(*j*)：返回索引 *i* 使 $SA[i]=j$。

在实际中使用压缩后缀数组的 3 个级别 0, $L'=\mathrm{lb}(\mathrm{lb}(n/2))$ 和 $L=\mathrm{lb}(\mathrm{lb}(n))$。$SA_L$的大小是$n/\mathrm{lb}n$。这里同样把$SA_L^{-1}$(即$SA$的逆函数)保存下来。该算法的伪代码如下：

```
int inverse(j)
{ e=loglogn/2; L=loglogn; k=0;
    while (k<L)   {
        r[k]=j mod 2e;   q[k]=j/2e;
        j= j/2e; k=k+e;
    }
    i=SAL-1[j]; k=L-e;
    while (k>=0)   {
        if (q[k]==0)   {
            i=posk1;
            for (d=1; d<r[k]; d++)
                i=Ψk[i]; }
        else   {
            i=Bk[i];
            for (d=0; d<r[k]; d++)
                i=Ψk[i];
        }
        k=k-e; }
    return i;}
```

其中，$SA_k[pos_k^{-1}]=n_k$，如果n_k不是$(\mathrm{lb}n)^{1/2}$的整数倍。

(2)search(P)：返回 *SA* 中的一个区间 [*l*, *r*] 使 $T[SA[i]\cdots SA[i]+|P|-1](1\le i\le r)$匹配 *P*。

这里希望能依据给出的模式 *P* 找到后缀数组中的区间 [*l*, *r*]。在文献[6]中提出的 *search* 函数的一个最大优点是不再需要原文本。*search* 函数依据如下：

$$T[SA[m]+i]=C^{-1}[\Psi^i[m]] \tag{3}$$

其中，C^{-1}是C的反函数，而$C[c]$等于所有小于c的字符出现次数的总和。

通过使用式(3)整个 search 函数的时间复杂度是 $O(|P|)$，这跟后缀数组没有压缩和保存原文本时是一样的。

(3)*decompress*(*s*,*e*)：返回子字符串 $T[s...e]$。

decompress(*s*,*e*)函数的功能是返回字符串 $T[s...e]$。该函数的伪代码如下：

```
decompress(s,e)
{ for (j=1; j<=m; j++)
    { S[j]=C-1[i];
        i=Ψ[i];
    }
    return S;}
```

增加了以上 3 个操作后，压缩后缀数组在搜索引擎技术中的应用会简单化。用 *search* 函数查找用户需要的模式，再用 *decompress* 函数得到原文本中的字符，从而在使用了压缩后缀技术后，就不再需要原文本了。

4 实验程序和扩展应用说明

用压缩后缀数组和倒排文件索引的方法对一组文件建立索引，测试 5 个用例字符串，其返回结果对比如表 2 所示。实验结果分析：使用压缩后缀数组时 $AVE(n)=31$，使用倒排文件则 $AVE(n)=25$。统计表明，短语的查准率提高了近 20%，而当短语退化为关键词时，两者的查准率一样。

表 2　压缩后缀数组索引与倒排文件索引的返回纪录比较

关键词	结果条数	
	倒排文件	压缩后缀数组
and	5	5
My cat	15	12
More and more	25	19
Xaut university	30	22
With your help	80	67

测试文本的内容(longdata.txt):pollution is becoming more and more serious all over the world. As is shown in the cartoon, two cars are giving off waste gas and three people trying to avoid breathing in the poisonous gas by masking their face with their hands. The poisonous gas sent off byfactories, domestic appliances and automobiles has made the air unhealthy for people to breathe.

说明：在构造程序输入该文件的路径和特殊字符 ($) 后，点击建立压缩后缀数组进行构建，并启动查询器(图 1)。图示输入了关键词 and，一共找到 4 个，在结果框中显示出了它们的字符位置编号。

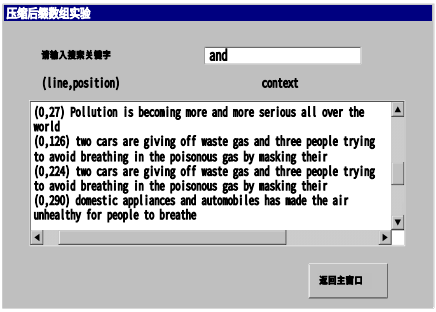

图 1　大文本关键字搜索(压缩后缀数组应用)程序 visio 图

使用压缩后缀数组技术，便于统计词条在文章中的出现次数，偏移位置等信息(上面的试验程序可以说明)，利用这些词条附属信息就知道该词条对于它所在文档的重要性，可以进行客观的权重设置，并存储到索引文件中。在生成结果列表时，通过比较查询词条权重的大小排序所得文章，用户便找到了与自己搜索关键词最相关的结果。同时可以设定一个阈值，当一个词条在一篇文章中的权重大于等于这个值时，就认定这个词条代表了这篇文章，确定为该文章的关键短语。这给文件自动归类提供了比较词汇，是文本聚类的数据基础。归类后，在生成的搜索结果界面中可以为用户提供类别信息，用户选择类别，结合负反馈和记录挖掘，不断优化搜索结果，步步逼近，最终找到用户所需。

图 2 中的程序实现了对 Intranet 内的常用文件格式(doc, ppt, pdf, xml 等)的文件搜索与共享下载，详情可访问博客 http://blog.sina.com.cn/timesoft。

图 2　基于 Lucene 与后缀数组的 Intranet 搜索引擎程序 visio 混合图

(下转第 88 页)