ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.20, No.7, July 2009, pp.1768−1784
doi: 10.3724/SP.J.1001.2009.03500

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

全文索引技术时空效率分析*

刘小珠[1,2], 彭智勇[3+]

[1](武汉大学 软件工程国家重点实验室,湖北 武汉 430072)

[2](武汉理工大学 自动化学院,湖北 武汉 430070)

[3](武汉大学 计算机学院,湖北 武汉 430072)

Time and Space Efficiencies Analysis of Full-Text Index Techniques

LIU Xiao-Zhu[1,2], PENG Zhi-Yong[3+]

[1](State Key Laboratory of Software Engineering, Wuhan University, Wuhan 430072, China)

[2](School of Automation, Wuhan University of Technology, Wuhan 430070, China)

[3](School of Computer, Wuhan University, Wuhan 430072, China)

+ Corresponding author: E-mail: peng@whu.edu.cn

Liu XZ, Peng ZY. Time and space efficiencies analysis of full-text index techniques. *Journal of Software*, 2009,20(7):1768−1784. http://www.jos.org.cn/1000-9825/3500.htm

Abstract: As one of the efficient methods of improving time and space efficiencies, the full-text index technique has been well studied in recent years. According to the implementation ways, it can be classified into three categories: Index technique, hybrid technique of index and compression, self-index technique. This paper reviews the recent researches on this topic, which include the main techniques such as inverted files, signature files, suffix trees, suffix arrays, compressed indices based on the indices mentioned above, self-index based on inverted files, and self-index based on suffix arrays. This paper also introduces the basic theories, problems, progress as well as space and time efficiencies of these techniques. Through a detailed efficiency analysis and comparison, the pros and cons of the techniques are given. Finally, the important features of these techniques are summarized, and the future research strategies and trends directions are pointed out as well.

Key words: inverted file; signature file; suffix tree; suffix array; self-index; compression; time and space efficiency

摘 要: 全文索引技术(full-text index technique)作为提高全文检索时空效率的有效方式之一,近年来得到了广泛而深入的研究.根据全文索引实现技术的不同,将其分为三大类:索引技术、压缩与索引混合技术以及自索引技术(self-index technique).从上述分类角度综述了全文索引时空效率方法中具有代表性的一些方法和技术:倒排文件、签名文件、后缀树与后缀数组、基于这 3 种索引的压缩技术、基于倒排文件的自索引与基于后缀数

* Supported by the National Natural Science Foundation of China under Grant Nos.60573095, 90718027 (国家自然科学基金); the National High-Tech Research and Development Plan of China under Grant No.2006AA12Z210 (国家高技术研究发展计划(863)); the National Research Foundation for the Doctoral Program of Higher Education of China under Grant No.20050486024 (国家教育部博士学科点专项科研基金)

Received 2008-03-21; Accepted 2008-10-07

组的自索引的基本原理、所面临的问题及进展,并对这些技术的时空性能进行了详细的分析和比较,分析了各种技术的适应环境及优劣.最后总结了上述技术的特点,指出了存在的问题以及未来的研究方向.

关键词: 倒排文件;签名文件;后缀树;后缀数组;自索引;压缩;时空效率

中图法分类号: TP311 **文献标识码:** A

据统计,在联机存储的信息中,80%以上的信息是以文本的形式存在的[1].如何快捷、有效地管理和检索文本这种非结构化的数据成为一项紧迫的研究任务,全文本信息检索(full-text information retrieval,简称全文检索)[2]技术由此应运而生.全文检索[2]是指以文本为检索对象,允许用户以自然语言的方式根据资料内容(例如文献中的任何一个词语)而不仅是外在特征(诸如标题、作者、摘要、附录等)来实现信息检索的手段,以支持多角度、多侧面地综合利用信息资源.全文检索技术的出现,导致了信息检索领域的一场革命.它不仅可以实现情报检索的绝大部分功能,而且还能直接根据数据资料的内容进行检索.

全文检索的研究可追溯至 20 世纪 60 年代[3].最初的全文检索是通过在文本中逐个字符扫描匹配完成的,不需要其他辅助数据.随着文本集越来越大,人们对全文检索的需求越来越多样,这种顺序比较方法效率低下的弊端就凸现出来.同时,由于早期受到计算机等技术的限制,既无法实现大量文本的存储,又不能实现快速的信息检索.因此,全文检索的时间与空间效率一直是人们衡量文本检索系统的关键指标之一.如何既能实现大量文本的有效存储,又能实现快速、准确的信息检索成为人们追求的目标.全文索引技术(full-text index technique)作为提高时空效率的有效方式之一,得到了广泛而深入的研究.经过近几十年的发展,人们在全文索引的时空效率研究中取得了很多实质性的成果,并得到了广泛的应用.根据实现技术的不同,本文将全文索引的时空效率研究方法分为如图 1 所示的 3 类.

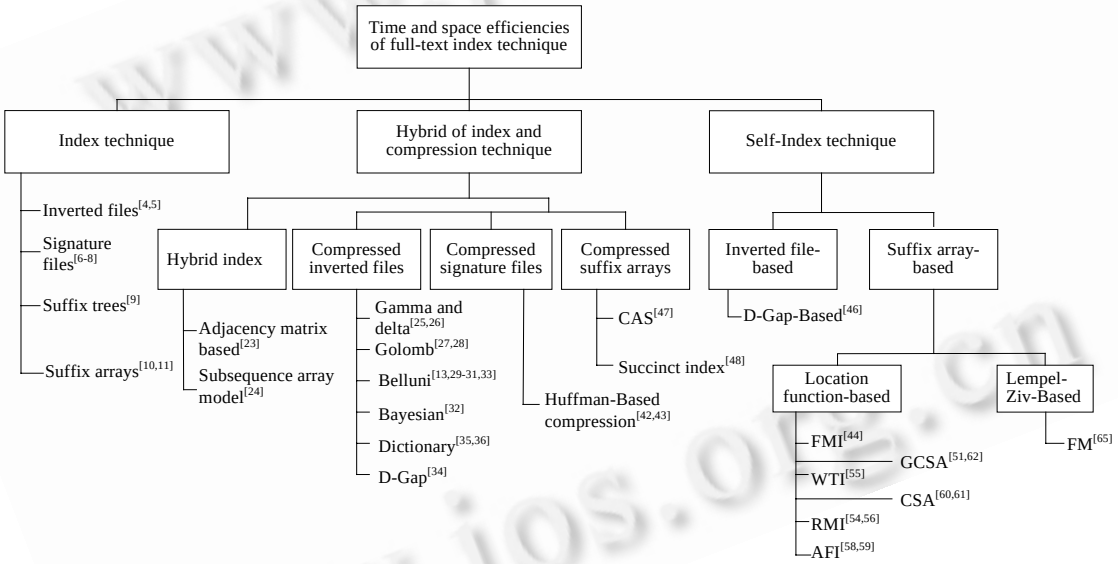

Fig.1 Classification of research methods on time and space efficiencies of full-text indexes

图 1 全文索引的时空效率研究方法分类

一是索引技术.人们受到书目索引的启发提出了全文索引的思想,通过存储辅助的索引数据,在适当地牺牲空间效率的前提下,显著地提高了全文检索的时间效率.并且提出了倒排文件[4,5]、签名文件[6−8]、后缀树[9]与后缀数组[10,11]等具有代表性、应用广泛的全文索引模型.

二是压缩与索引混合技术.一方面,通过将各种索引技术的优点相结合实现混合索引技术以提高全文检索的性能.另一方面,随着计算机技术的发展,人们将各种数据压缩技术引入全文检索系统中,在适当地牺牲时间效率的前提下,不仅有效地实现了海量文本信息的存储,而且实现了高效的全文检索.

三是自索引(self-index)技术.通过充分地发展压缩索引技术,提出了使时空效率同时提高成为可能、完全不依赖于原文本的新一代索引技术——自索引.自索引技术从一定程度上改变了传统索引技术与压缩技术在时间效率与空间效率上相对立的矛盾.

本文着眼于全文索引技术时空效率的研究,从上述分类角度综述了全文索引时空效率方法中具有代表性的一些方法和技术,并对相关技术的性能进行比较.重点讨论了索引技术、压缩与索引混合技术以及自索引技术的基本原理及进展,从相关问题、现状和趋势等方面进行归纳和评论.第 1 节介绍 3 种具有代表性的索引技术并对 3 种技术的时空性能进行对比分析.第 2 节讨论混合索引技术、压缩与索引混合技术并对 3 种压缩索引技术的性能进行比较分析.第 3 节介绍两种自索引技术,并对这两种自索引的性能进行比较分析.最后对当前工作存在的问题进行分析与总结,并展望相关技术的未来发展.

1 索引技术

索引(index)最早出现在文献系统中,从这个意义上讲,索引是指文献集合中包含的事项或从文献集合中引出概念的一种系统指南,这些事项或引出的概念是按已知的或已说明的可检索顺序排列的条目表达出来的.如第 1 个专门为圣经编制的索引、带索引的书籍、医学文献索引目录等.以上索引的提出,为现代索引技术的发展奠定了基础,具有深远的影响.随着计算机的出现,索引技术得到了迅猛的发展,尤其是数据库系统中索引技术的研究,长期以来都得到高度重视,并提出了很多应用广泛的索引.但是传统数据库擅长于结构化数据的管理,其索引是关键字索引;而文本是典型的非结构化数据,文本检索要求针对全文建立索引,并且还要求具有相关度排序、高亮显示等附加功能.这些差异使得全文检索系统的实现方式以及全文索引的结构与传统数据库截然不同,因此无法通过对传统数据库技术的移植、借用和变换等简单方法来实现全文检索,必须寻求全新的理论和方法.

全文检索的关键技术是索引的建立与维护,本文重点讨论索引的建立机制,对于索引的维护与更新机制将在后续论文中加以论述.目前具有代表性的索引技术有倒排文件、签名文件以及后缀树与后缀数组等.

1.1 倒排文件

倒排文件(inverted file)[4,5]是从书目索引中受到启发而派生出来的,它是目前应用最广泛的全文索引模型.倒排文件机制是一种面向单词的索引机制,利用它可以提高检索速度.倒排文件主要分为以下两种.

(1) 文档级倒排文件(document-level inverted file)

在文档级倒排文件中,索引由两部分组成:第 1 部分是词表.包含两部分信息:一是包含词 t 的文档数量 f_t,二是指向文档倒排列表的指针.第 2 部分是倒排列表 L_b.包含两部分信息:一是包含词 t 的文档的标识(identifier,ID)号 d,二是文档 d 中词 t 出现的次数 $f_{d,t}$.倒排列表由文档的 ID 号 d 与文档 d 中词 t 出现的次数 $f_{d,t}$ 组成的$\langle d,f_{d,t}\rangle$序列对构成.

(2) 单词级倒排文件(word-level inverted file)

由于文档级倒排文件不包含词在文档中出现位置的具体信息,导致搜索效率不高.为了提高搜索效率,单词级倒排文件对索引中的第 2 部分倒排列表 L_b 进行了改进:在 L_b 中加入了词在文档中出现的位置信息 p_i,即$\left\langle d, f_{d,t}, p_1, p_2, ..., p_{f_{d,t}} \right\rangle$,可以实现精确的定位,避免在原始文本中搜索的时间开销.

在倒排文件中,对于有 n 个字符的文本,词表对空间的需求相对较小,其增长率为 $O(n^{\beta})$[12],其中β为[0,1]的常量,在实际系统中,其值一般为 0.4~0.6 之间.对于 1GB 的文本信息,词表的大小约为 5MB.索引中倒排列表的存储空间为 $O(n)$,索引的创建可以在 $O(n)$的时间内完成.在实际应用中,由于存放倒排列表时还必须存放一些描述信息,因此,倒排列表的存储空间变化较大,可能会达到原文件大小的 300%[12].

1.2 签名文件

签名文件(signature file)作为一种文本检索技术也被称为散列函数法[6−8].在签名文件中,每个文档都通过 Hash 函数及重叠编码(superimposed coding)产生一个称为目标签名(target signature)的位串.具体做法是:将文档

中的每个词作为散列函数的参数产生二进制词签名(w位的向量),当把文档中每个词的词签名叠加时,就得到了合并文档的目标签名.文档的目标签名结果顺序存入一个单独的文件(签名文件)中.由于签名文件采用的是二进制的叠加,比原文件小得多,因此可以提供快速的检索.当数据对象较大时,为了提高效率,可将其分成若干个逻辑块,每个逻辑块产生相应的目标签名.

由于签名文件方式的产生采用了类似于二进制的编码方式,因此签名文件占用的空间很小,一般为原文本集大小的30%~50%[13].由于签名的产生方式决定了签名文件的大小及查询匹配的效率与正确性,为了提高签名文件的时空效率,很多研究工作都集中在签名文件的设计方式上,并提出了很多应用广泛的签名文件[14−17]用以进一步提高检索性能.

1.3 后缀树与后缀数组

后缀树(suffix tree)是一种特殊的 Patricia 树[9],其最大的特点是将一个文本看成一组半无限串的叠加,而这组半无限串的排序结果被表示成树的形式.

对于有 n 个字符的文本 $T_{1,n}$,后缀树的空间效率为 $O(n)$,创建时间为 $O(n)$[9,18,19].其存储空间为 $O(n\log_2 n)$比特,后缀树的存储消耗巨大,在实际应用中至少是原文本大小的 10 倍[20].

后缀数组(suffix array)由 Gonnet,Manber 和 Myers[10,11]在改进后缀树模型的过程中提出.他们将后缀树的叶节点串行化就得到了后缀数组.后缀数组是文本 $T_{1,n}$ 所有的后缀按照字母大小顺序排序后的简单排列.对于文本 $T_{1,n}$,其后缀数组为包含范围为[1,n]的排列的一个数组 A[1,n],该数组对于所有的 $i\in$[1,n]有 $T_{A[i],n} < T_{A[i+1],n}$.其中"<"为按照字母顺序的大小关系,$A[i]$表示后缀数组中第 i 个元素对应到原始文本中后缀开始的位置.

对于有 n 个字符的文本 $T_{1,n}$,后缀数组的存储空间消耗为 $n\log_2 n + n\log_2 \sigma$ 比特[10],当搜索文本为 $S_{1,m}$ 时,搜索时间消耗为 $O(m\log_2 n)$,文本定位时间消耗为 $O(1)$,显示 l 个字符的时间消耗为 $O(l)$.这里,σ为文本中字符所在字符集的所有字符的个数.

当用更多的空间存储连续的后缀数组之间的最长相同前缀信息时[10,21],对于有 n 个字符的文本 $T_{1,n}$,后缀数组的存储空间消耗为 $2n\log_2 n + n\log_2 \sigma$ 比特,当搜索文本为 $S_{1,m}$ 时,搜索时间消耗为 $O(m+\log_2 n)$,文本定位时间消耗为 $O(1)$,显示 l 个字符的时间消耗为 $O(l)$.

1.4 3种索引的性能分析与比较

3 种索引的性能比较见表 1.

Table 1 Performance comparison of three kinds of indexes

表 1 3 种索引的性能比较

em	Inverted file	Si nature file	Suffix array
Space cost	50%~300%	30%~50%	>>100%, $n\log_2 n+n\log_2 \sigma$
Index construction	Slowly,$O(n)$	Slowly	Slowly
Updating cost	Medium,<$O(n)$	High	High
Retrieval cost	Medium, $O(\log_2 n)$ +Merging time	Medium	Low, $O(m\log_2 n)$
Ranking	Yes	No	No
Scalability	Sublinear	Linear	Linear
Extensibility	Yes	No	No

倒排文件有很多优点,如实现相对简单、查询速度快、很容易支持同义词查询以及排序查询等扩展功能.但是倒排文件也存在很明显的缺点[22]:由于倒排列表是以排序的方式进行存储的,因此在查询估计过程中对倒排列表进行排序的开销巨大;倒排列表的存储空间变化较大,导致倒排文件的大小范围可以从原文本大小的50%增加到 300%[12];索引创建开销较大,动态环境下索引文件更新和重新组织的开销与索引合并的开销要大;对于有 N 个文档的文本集,倒排列表的磁盘访问次数为 log$N$,因此磁盘访问开销大.

如表 1 所示,签名文件的最大优点是空间效率高,由于其二进制特性,签名文件形成的索引存储结构紧凑,其大小一般为原文本集大小的30%~50%[13].但是存在误匹配是签名文件的一个主要缺点.尽管可以通过加大签名宽度 w 来降低误匹配率,但这将增加索引空间的大小.并且在查询过程中,由于签名文件的特性与数据集的多样

性导致误匹配不可避免,这在一定程度上增加了查询的开销.另外,签名文件不支持排序查询,同时,由于其二进制特性难以实现功能的扩展.

后缀数组的最大优点是极大地加快了检索速度,尤其是对某些特殊的检索,如前缀检索、范围检索等检索效率更高.它的最大缺点是索引生成复杂、空间开销大,而且创建过程中的空间开销更大,创建效率也很低.此外,无论是创建过程还是检索过程都严重依赖原文本.后缀树和后缀数组在创建和合并的过程中均需要大量移动数据,因此,两者的动态性能都不理想.另外,后缀数组与签名文件一样,不支持排序查询,扩展性较差.

从空间开销上看,由于签名文件的二进制特性使其最具优势,而后缀数组由于重复存储了文本中的所有字符,使其空间开销最大,达到 $n\log_2 n+n\log_2\sigma$ 比特,其大小一般是原文本的 10 倍;由于索引所占的空间大、存储结构紧凑,使得索引文件更新和重建的开销增加,这样不仅增加了磁盘的访问量,也增加了对内存空间的需求.因此,3 种索引的创建开销都较大;在索引更新开销上,倒排文件由于支持部分更新,其性能略有优势,而签名文件的二进制特性、后缀数组的复杂结构使其只能采用完全重建索引的方式实现更新,因此更新效率很低,尤其是对于信息频繁更新的应用,这种完全依赖于重建的索引很难满足用户需求;后缀数组在检索性能上最具优势,时间消耗为 $O(m\log_2 n)$,检索时间消耗低于倒排文件与签名文件.另外,倒排文件很容易实现排序查询,而签名文件与后缀数组因为结构紧凑难以支持排序查询;在可伸缩性上,对于给定的查询,3 种索引对磁盘的访问次数取决于从索引中读取的数据量:对于倒排文件,其索引处理消耗与数据的大小呈次线性关系,而签名文件与后缀数组的索引处理消耗与数据大小呈线性关系;倒排文件由于存储的信息全面,表现出了良好的可扩展性,而签名文件与后缀数组由于索引本身并不包含数据记录中的某些信息(如文档结构、词频等),难以快速实现索引与原始信息的对应,因此无法实现位置信息查询、文档结构查询等功能.

可以看出,上述 3 种索引的性能没有非常明显的差别,都有各自的优缺点与适用性.但在时间与空间效率上,3 种索引技术也都有很大的提升空间.

2 压缩与索引混合技术

2.1 混合索引技术

为了提高时空性能,研究者对倒排文件、签名文件与后缀数组 3 种索引的混合索引技术进行了研究.混合索引的思想是通过将上述几种索引有机地结合,发挥各自的优点以提高索引的时空效率.文献[23]中提出了两种全文索引模型:基于邻接矩阵的倒排文件和基于邻接矩阵的后缀数组,用于改善信息检索系统的查询效率.该方法通过使用有向图表示文本串,引出文本串的邻接矩阵,并给出了基于上述两种模型的文本查询算法.提出的模型能够以相对于原文较小的空间代价获得较大幅度的查询效率提高,因此适合于大规模文本检索系统.刘学文等人结合倒排文件与后缀树的优点,提出了一种改进的后继数组模型[24].传统后继数组模型的出发点是:建立了一个文本,实际上就确立了一组字符的排列顺序(排列),文本文件用于记录、存放该排列.另外,当对一个文本建立了后缀数组索引之后,这组字符又确立了一个排列顺序,后缀数组索引文件记录、存放该排列.这样,文本中的每个字符同时在两个排列中担任角色,而且这两个排列存放在不同的文件中,而在创建和使用过程当中都要频繁穿插这两个排列(查询时使用后缀数组排列,读取和显示时使用文本排列),因此导致频繁的文件打开(关闭)操作,继而导致系统效率降低.该文提出的后继数组模型将这两种排列保存在一个文件中,以减少文本位置信息的存储,提高时空效率.

混合索引技术在一定程度上提高了全文检索的性能,但是这些方法存在以下问题:一是对全文检索性能的改善非常有限,当面对海量的文本数据时,其性能并不理想,因此还有待于进一步完善;二是受所联合的索引技术固有缺陷的限制,很难用一种索引的优势去弥补另一种索引的不足,因此难以形成有效的索引;三是这些貌似综合了多种索引优点的混合方案往往将索引的创建过程和检索过程变得更为复杂,混合过程中增加的复杂度抵消了其带来的性能改进,并不能达到提高性能的目的.因此必须研究新技术提高全文检索的时空性能.

2.2 索引压缩技术

早期的检索技术由于受计算机硬件的限制,既无法实现大量文本信息的存储,也无法实现信息的快速检索.随着计算机技术的发展,数据压缩技术作为海量信息存储和传输的支撑技术得到了人们极大的重视,并被引入到全文信息检索中.采用一定的压缩策略,不仅可以节约存储空间,还可以减少查询时读取索引数据所需要的磁盘 I/O 次数,从而提高检索速度.因此,索引压缩技术作为提高全文检索时空效率的核心技术,得到了广泛而深入的研究:一是对正文进行压缩,即对文本数据进行压缩,实现大规模文本数据存储;二是对索引进行压缩,即对整个文本数据库建立的索引进行压缩,通过对索引进行高效的压缩编码以提高全文检索性能.由于正文压缩技术的目的是实现海量信息的存储,这里不作详细讨论.下面将重点讨论索引压缩技术.

2.2.1 倒排文件压缩

为了缩小倒排文件的存储空间,可以对倒排文件进行适当的压缩.目前,倒排文件压缩技术主要采用整数编码,即采用变长比特实现整数表示,可分为无参码与有参码两种方式.

(1) 无参码.编码本身隐含了整数序列中数值的概率分布,这种隐含概率分布的特性使其不但能够用于静态整数序列的压缩,而且也适用于动态整数序列的压缩.

如最简单的一元码(unary),采用 x–1 个比特的 1 表示 x,用 0 来结尾.一元编码并不节省空间,因此只是被用作其他编码的前缀.由于倒排列表可能很长,并且信息检索受到磁盘的限制,从磁盘上读取倒排列表的时间比解压该列表所需要的时间更多.为了提高检索效率,需要一种更有效的压缩方法来提高磁盘的吞吐量.Elias 提出了两种有效的索引压缩方法:Gamma 编码和 Delta 编码[25].后来,Bentley 和 Yao 作了更详细的描述[26].Gamma 编码将整数 x 编成由两部分组成的代码,前一部分是 $1+\lfloor \log_2 x \rfloor$ 的一元编码,后一部分是 $x-2^{\lfloor \log_2 x \rfloor}$ 的二进制编码,其位数为 $\lfloor \log_2 x \rfloor$ 比特.Delta 编码的前面部分为 Gamma 编码,后面部分是 $\lfloor \log_2 x \rfloor$ 的二进制编码.

(2) 有参码.对所有的倒排列表使用统一的压缩参数,假定已知每个数值的出现概率,就可以使用该概率以贝努里实验对数值出现进行模型化,其代表为 Golomb 码[27].Golomb 编码是在 1966 年被首次提出的,Gallager 和 Van Voorhis[28]在 1975 年发展了这种方法.Golomb 编码方式将整数 x 编成由两部分组成的代码.对参数 b,第 1 部分以一元编码的形式对 q+1 进行编码,其中 $q=\lfloor (x-1)/b \rfloor$,第 2 部分是 $r=(x-1)-q\times b$ 的二进制编码.如何选择 b 值是 Golomb 编码的关键问题,不恰当的 b 值会造成较长的编码及较长的解压时间.

由于 Gamma,Delta 编码与 Golomb 编码压缩性能高,在实际系统中得到了广泛的应用.除上述编码方式外,研究者针对各种特定的应用提出了很多有效的编码方式:如具有可变参数编码适合于文本文件的位图矢量变形贝努里模型[29−31],该方法提高了倒排文件的检索效率,但是这类方法由于编码参数选择需要增加统计功能而使编码的复杂性增加,同时,位图矢量占用的空间也有所增加;为了减少位图矢量占用空间,在上述工作的基础上,基于 Bayesian 编码[32]提出了改进的马尔可夫模型,该模型以 Bayesian 编码方式对数据库中的词进行统计压缩,以适当的编码复杂性为代价较大地提高了空间效率;采用分层次压缩方式提出来的局部贝努里模型[13,33],减小了倒排文件的占用空间,适合于图像与文档文件;采用基于整数的间隙编码(d-gap)[34]方式提出的适合于文本的倒排文件压缩技术,通过插入辅助信息进行快速定位以减少压缩数据的解压过程,提高了检索效率;为了进一步提高检索效率,基于压缩字典[35,36]的方式提供了在压缩的索引上直接支持搜索的功能,但是该方法的缺点在于:一是采用附加部分数据的方式提高检索时间效率是以牺牲空间效率为前提的;二是附加数据的编码在一定程度上增加了实现的复杂性以及解码时间.

在索引动态性上(即压缩索引在动态更新与查询时是否需要先解压后才能实现索引的调整与查询),无参码如 Gamma 编码、Delta 编码、可变字节编码等都具有良好的动态性.其中可变字节编码的压缩率较低,而且由于该编码采用字节对齐方式,能够充分发挥硬件的速度,压缩和解压的时间效率具有显著的优势;Gamma 编码只适合压缩很小的整数;尽管 Delta 编码在表示小数值时所占用的比特数比 Gamma 编码要多,但对于较大的数所占用的空间情况则正好相反.总体而言,Delta 编码在解压速度与压缩率上优于 Gamma 编码.

有参码包括 Golomb 编码及各种变形贝努里模型,其编码取决于某些参数,而这些参数又与概率分布相关.

当整数序列动态变化时,会导致整数概率分布的变化,从而导致其参数的变化,最终使所有整数的编码都要发生改变,因此,带参数的编码都不适合对动态序列的压缩.Golomb 编码与 Delta 编码、Gamma 编码相比,不仅编码速度更快,而且压缩率更高.

研究表明[5,13],对于 1GB 的文本数据,在 Solaris 系统、Sun SPARC10 处理器及 256MB 内存的系统配置下,将每个实验执行 5 次取其平均值以减少外部因素的干扰,采用参数 b 取 5 时的 Golomb 编码.通过对倒排文件的倒排列表、索引词位置信息等进行压缩后与不采用压缩时相比:存储空间减少了 5 倍、内存空间消耗减少了 20 倍、CPU 资源消耗减少至少 3 倍、I/O 操作减少 5 倍、索引创建时间至少降低 2 倍.这使得倒排索引的时空效率得到了极大的提高,压缩的倒排索引得到了广泛的应用.

2.2.2 签名文件压缩

由于签名文件的产生方式及签名宽度 w 的选择对签名文件的性能起着决定性的作用,因此,对签名文件的研究大多都集中在签名的产生方式上.如对文本进行分块以减少签名的宽度,使查询的消耗与文本集的大小呈线性关系[17,37−39]等.但是这些工作还存在两个问题:一是这些方法并没有从根本上解决签名文件存在的误匹配(false match)问题;二是这些方法都基于单词的独立概率假设来设置签名的比特矢量,这与实际文本集中的单词分布情况相去甚远[40,41].上述问题再加上签名文件的二进制特性,使得签名文件的压缩变得困难.因此,有关签名文件压缩的研究并不多见.而且,现有研究工作大都集中在比特分段签名文件的压缩上,如文献[42,43]中提出的压缩方法,其思想是,根据目标签名中连续 0 与连续 1 的个数,采用改进的 Huffman 编码方式实现目标签名的压缩.但是,该方法依赖于签名的宽度 w 以及签名中 1 的分布情况.当签名矩阵为稀疏矩阵时(1 的分布),其压缩性能较好;但对于其他情况,其性能并不理想.另外,由于解压缩带来的时间消耗也使得查询效率变得更低.根据目前的研究,签名文件及其压缩技术存在以下问题:

(1) 当每个数据对象中所包含的单词数变化范围较大时,导致目标签名宽度 w 变化较大,而在签名文件中存储的目标签名必须具有相同的签名宽度,这将导致签名宽度 w 难以确定,这已成为基于签名文件的文本检索中不能忽略的严重问题:签名宽度 w 的选择直接影响索引的空间大小与误匹配的概率.

(2) 常用词(高频词)的实际分布使得签名处理需要额外的开销.当高频词与生僻词所产生的词签名进行重叠编码形成目标签名时,此时对于生僻词查询产生误匹配的概率随高频词的分布概率增大而有所增加.

(3) 签名文件的参数,如每个词的比特数、签名宽度、块签名因子难以设定.当参数选择不合理时,将导致查询速度与空间效率性能降低.另外,对于某一数据集查询有效的参数设置并不一定适合于其他数据集.

(4) 当对签名文件采用压缩技术时,上述因素导致了压缩性能的不确定性.另外,由于签名文件所具有的二进制特性,使得签名文件的压缩变得困难.尽管采用适当的压缩技术可以使签名文件所占有的空间大小降低一半,但解压缩的时间消耗抵消了其带来的性能改进,查询效率将变得更低.

2.2.3 后缀数组压缩

后缀数组的压缩是通过对定位函数进行间隙编码(gap coding)[33]来实现的.定位函数是在进行搜索时,后缀数组对搜索文本 $S_{1,m}$ 进行匹配定位所满足的关系式.目前,后缀数组的搜索方法分为反向搜索(backward search)和正向搜索(forward search)两种.因此后缀数组的压缩可以分为基于反向搜索的压缩与基于正向搜索的压缩两种方式,其原理都是对定位函数进行间隙编码.

反向搜索[44]的核心思想是:对搜索文本 $S_{1,m}$,先从最后一个字符进行匹配搜索,然后再匹配最后两个字符,直到搜索到与搜索文本 $S_{1,m}$ 中所有字符都匹配的字符串为止.而正向搜索的顺序与反向搜索相反,即从 $S_{1,m}$ 的第 1 个字符开始匹配,直到最后一个字符.

反向搜索算法的特点在于基于后缀数组实现了搜索字符的快速定位.如果用 $LF(i)$表示定位位置,L_i 表示 $T_{A[i]-1}$,即当前后缀的前一个字符列的第 i 个元素,$C[L_i]$表示在后缀数组 $A[1,n]$中比字符 L_i 小的字符数,$Occ(L_i,i)$表示在 $T_{A[i]-1}$ 列中,从 1 到 i 行中字符 L_i 出现的次数,则反向搜索算法有如下关系式成立[44,45]:

$$LF(i)=C[L_i]+Occ(L_i,i)=i' \tag{1}$$

$$A[LF(i)]=A[i']=A[i]-1 \tag{2}$$

从式(1)和式(2)可以看出:$LF(i)$实现了后缀 $T_{A[i],n}$ 到 $T_{A[i]-1,n}$ 的映射.

如果定义 $LF(i)$的反函数为 Ψ,则可实现后缀 $T_{A[i],n}$ 到 $T_{A[i]+1,n}$ 的映射,即可以实现文本 $T_{1,n}$ 从左到右地搜索,即正向搜索.反函数Ψ的特点在于,对于后缀 $T_{A[i],n}$,如果已知当前字符 $T_{A[i]}$在后缀数组的位置索引 i,则有 $\Psi(i),\Psi(\Psi(i))$等分别指向字符 $T_{A[i]+1}$,$T_{A[i]+2}$ 等,即能够快速定位后续的字符,只需要经过 n 步的计算,就可以获得 $T_{A[i]}$之后的 n 个字符,即 $T_{A[i],n}$.

无论是正向搜索还是反向搜索,其压缩的原理相同.其核心思想是对定位函数或其反函数进行适当的编码,如采用间隙编码进行压缩,并辅助存储一定的信息数据,如 $C[L_i]$值等,在压缩数据上直接支持快速的查找功能,进而减少其他信息的存储,提高空间效率.

对于定位函数的反函数 Ψ,由于 Ψ的值为一系列整数,例如,对于 $\Psi(i)$={4,17},采用 d-gap 的压缩方式[34],即

$$d_i = d_{i+1} - d_i,\ \ i>1 \tag{3}$$

则有 $\Psi_d(i)$={4,13}.

为了在压缩数据上直接支持快速查找与定位 Ψ值的功能,一种简便但并不十分有效的方法是,每隔一定数量的压缩码之后插入一个指针指向下一个具有绝对值的压缩码的地址.当需要查询某个 Ψ值时,先找到最近的具有绝对 Ψ值的地址指针,然后通过间隙值相加进行解码,直到找到所查找的对象.这种思想与文献[46]中提出的基于倒排文件的自索引是一致的.这种方式中,最多需要对 $\log_2 n$ 个值进行解码,因此任一 Ψ值的计算时间消耗为 $O(\log_2 n)$,插入的附加信息的空间消耗为 $O(n)$比特.通过进一步地改进编码方式,在压缩数据上直接快速查找与定位 Ψ值的时间消耗可以减小为固定的常数.

基于上述原理,在文献[47]中提出了一种紧凑的后缀数组 CSA(compact suffix array).CSA 通过结构化的压缩方式,将后缀数组的存储空间最小化.CSA 的基本原理是采用自重复(self-repetition)的方式替代后缀数组中重复的部分,极大地提高了后缀数组的空间效率.Grossi 等人[48]提出了一种压缩的后缀数组——一种简洁索引(succinct index),即在不存储后缀数组的前提下,提供查询与访问后缀数组的功能.其思想是,基于定位反函数 Ψ,采用分等级的解压缩方式实现后缀数组的解码.简洁索引不仅改变了后缀数组对原始文本的依赖特性,同时从一定程度上提高了压缩后缀数组的空间效率.但是这两种索引的缺陷在于,索引创建过程复杂,查询时间效率还有待于进一步提高,尤其是无法支持索引的部分更新、排序查询等功能.

2.3 3种压缩索引的性能

对于同一个 1GB 的数据集,倒排文件与签名文件压缩后的性能比较见表 2.从表 2 中可以看出,无论是空间效率,还是时间效率,倒排文件的性能都优于签名文件.在存储空间上,倒排文件的存储空间低于原文本大小的 15%[22,49],与签名文件相比至少降低 1 倍存储空间;在索引创建时间效率上,倒排文件需要的时间为 40 分钟,而签名文件则需要 86 分钟;尽管在索引更新的开销上,两种技术的性能相当,但倒排文件支持部分更新,而签名文件难以支持新记录的随时加入.另外,压缩后的倒排文件处理消耗与数据大小呈次线性(sublinear)关系,而且支持排序查询、位置信息插入、文档结构查询、分离查询等扩展功能,这都表明压缩倒排文件具有良好的可伸缩性与可扩展性.而签名文件处理消耗与数据大小呈线性关系,难以支持排序查询,且可扩展性差.因此,通过压缩技术,倒排文件的性能得到了进一步提高,与签名文件相比优势更加明显.

Table 2 Performance comparison of compressed inverted file and compressed signature file

表 2 倒排文件与签名文件压缩后的性能比较

Item	Inverted file	Signature file
Space cost	<15%	30%~55%
Index construction	40 minutes	86 minutes
Updating cost	45%	45%
Retrieval cost	Medium	Medium
Ranking	Yes	No
Scalability	Sublinear	Linear
Extensibility	Yes	No

倒排文件与后缀数组压缩后的性能相比,同样具有很多优势,见表 3.

存储空间:倒排文件空间效率高,存储空间大小低于原文本大小的 15%;而后缀数组尽管采用压缩技术提高了空间效率,存储空间大小至少为原文本的 1 倍以上,但与倒排文件相比,其性能仍有较大差距,而且后缀数组在压缩时需要存储定位函数等附加信息,从而增加了额外的空间开销.

Table 3 Performance comparison of compressed inverted file and compressed suffix array

表 3 倒排文件与后缀数组压缩后的性能

Item	Inverted le	Suffix array
Space cost	<15%	>100%
Index construction	Medium	High
Updating cost	<45%	High
Retrieval cost	Medium	Low
Compression complexity	Low	High
Location cost	Medium	Low
Display cost	Medium	Low
Ranking	Yes	No
Scalability	Sublinear	Linear
Extensibility	Yes	No

索引创建:倒排文件的索引创建采用复杂性低的整数编码,因此创建速度快;而后缀数组的创建速度一方面依赖于所采用的压缩算法的复杂性,另一方面,为了实现由索引直接还原原始数据,必须附加很多其他信息,这增加了索引创建的开销.

索引更新维护:倒排文件支持索引的更新与维护,能够实现部分更新与索引重建,而且开销低于 45%;但对于后缀数组,当有新的数据加入或数据发生变化时,目前的技术还无法实现部分更新,只能对整个后缀数组进行完全重排,并对定位函数进行重算等.因此难以实现索引的动态更新,尤其是对于大数据集,需要对后缀数组进行完全重构,索引维护更新成本更高.

排序查询:倒排文件支持排序查询功能且易于扩展,如支持语义相似度、文档结构查询与分离查询等功能;后缀数组由于其紧凑的结构特性,只能完成完全匹配时的查找定位,难以支持相似度匹配、排序查询等功能,且难以实现位置信息插入、文档结构查询、分离查询等功能.

查询速度与定位开销:倒排文件的查询速度虽然有了很大提高,但是由于后缀数组采用基于定位函数的匹配方式,能够快速计算出查询词的位置信息,因此与倒排文件相比,后缀数组在信息查询速度与定位开销上具有明显的优势.

3 自索引(self-index)技术

索引与压缩技术的结合,一定程度地提高了索引的空间效率:压缩索引减少了索引的空间消耗,读取和解压一个已压缩的索引比读一个未压缩的倒排索引更能节省磁盘访问时间,有助于提高查询的吞吐量.先前的研究关注得更多的是压缩率,而往往忽略了索引的动态性,即压缩索引在查询、动态更新时是否需要先解压后才能实现相应的功能.而压缩率与动态性往往是相矛盾的,压缩带来的解码时间消耗,在一定程度上降低了索引的时间效率.自索引技术一定程度地克服了上述矛盾,从而使时空效率同时提高成为可能.目前的自索引可分为两种:基于倒排文件的自索引以及基于后缀数组的自索引.

3.1 基于倒排文件的自索引

Moffat 与 Zobel 在文献[46]中提出了一种基于倒排文件的自索引,该索引通过在倒排文件列表中插入少量的忽略信息(skipping information)来提高倒排列表的时间效率.

3.1.1 自索引的形成

基于倒排文件的自索引的基本思想是:采用间隙编码方式[34]对文档号进行压缩.对索引列表中文档号 d 与词出现的次数 $f_{d,t}$ 形成的$\langle d,f_{d,t}\rangle$序列对的集合,先对文档号进行 d-gap 压缩,然后将$\langle d,f_{d,t}\rangle$序列对以一定的数量进行分组,插入忽略信息$\langle d_i,a_i\rangle$.进一步将插入的忽略信息$\langle d_i,a_i\rangle$中的地址 a_i 采用 d-gap 压缩以提高空间效率.此时对该序列采用适当的压缩技术,如 Golomb 进行编码,即形成了基于倒排文件的自索引.

当要检索压缩列表中是否包含文档 d 时,首先对第 1 个忽略信息进行解码得到$\langle d_1,a_1\rangle$,并根据地址 a_1 得到第 2 个忽略信息的物理地址,然后对第 2 个忽略信息进行解码得到 $\langle d_2,a_2-a_1\rangle$.对于文档 d,如果 $d_1\le d<d_2$,则表明该文档位于第 1 分组中,只需要将第 1 分组进行解码即可.否则,如果 $d_2\le d$,则需要根据 $\langle d_2,a_2-a_1\rangle$ 中的地址信息得到第 3 个忽略信息的物理地址,进行解码获得 $\langle d_3,a_3-a_2\rangle$,再重复采用上述方法,直到找到文档 d 所在的压缩分组,并进行解码即可.

3.1.2 自索引的效率

对于有 $p=|I_t|$ 个 $\langle d,f_{d,t}\rangle$ 序列对的倒排列表,当有 $k=|C|$ 个文档时,假设每个 $\langle d,f_{d,t}\rangle$ 的解码时间为 t_d,倒排列表中的忽略信息个数为 $2p_1$,当考虑 CPU 将数据读入内存的时间消耗时,如果用 t_r 表示将一个 $\langle d,f_{d,t}\rangle$ 读入内存的时间,则搜索一个倒排列表所需要的时间 T 及其最小值 $T_{\min}$ 为

$$T=t_d\left(2p_1+\frac{kp}{2p_1}\right)+t_r(p+2p_1) \tag{4}$$

$$T_{\min}=2\sqrt{t_d(t_d+t_r)}\cdot\sqrt{kp}+t_rp \tag{5}$$

将上述方法加以扩展,采用递归的方式对插入忽略信息的列表进行索引,再插入忽略信息,可以进一步提高时间效率.对于 h 级的忽略插入及同步点 $p_1,p_2,...,p_h$,处理时间的最小值 $T'_{\min}$ 为

$$T'_{\min}=t_d(h+1)k^{\frac{h}{h+1}}p^{\frac{1}{h+1}}+t_r\left(p+2\sum_{i=1}^{h}p_i\right) \tag{6}$$

实验结果表明[46],对于 2GB 的文本数据,采用自索引技术与不采用自索引技术相比,对于 5~10 个词的连接布尔查询,处理时间减小约 5 倍;对于排序查询,处理时间减小约 2~4 倍.而插入忽略信息的大小约为 0.04GB,即原始文本大小的 2%.通过采用递归的方式对插入忽略信息的列表进行索引,并采用压缩技术,可以同时减小内存空间、存储空间与处理时间.这种基于倒排文件的自索引,使时空效率同时提高成为可能.

基于倒排文件的自索引的本质是以插入少量的辅助信息为代价,尽可能地减少压缩带来的解码时间消耗,即通过辅助信息在不需要完全解码的情况下,实现信息的准确定位与查询功能.这种索引不仅具备复杂性低、简单且易于实现的优点,同时具有良好的查询性能和可扩展性,但其缺陷在于:一是当倒排文件中的位置信息列表被分成大量的小块时,插入的辅助信息不仅会导致大量的额外存储开销,同时由此产生的磁盘访问时间将超过压缩所带来的性能;二是难以同时支持连接布尔查询与排序查询;三是这种倒排文件自索引在文本显示功能上仍然依赖于原文本.

3.2 基于后缀数组的自索引

基于后缀数组的自索引[50]不仅具有后缀数组查询速度快、定位开销小的优势,而且包含足够的信息,能够有效地实现任意子字符文本的重构,从而达到不依赖原始文本的功能.

3.2.1 自索引的形成

基于后缀数组的自索引的基本原理如下:

一是定位函数实现了搜索字符的快速定位,有效地提高了时间效率;而利用小波树[51]能够有效地解决二进制序列上以空间为 $\log_2\sigma$ 的排序查询的特性,将每个 $Occ(c,i)$查询的存储空间从σn 降低到 $n\log_2\sigma$,实现空间效率的提高.而将后缀数组与小波树相结合可以同时提高时空效率.

二是基于定位函数的搜索算法实现了搜索字符的快速定位,但是搜索字符的定位与显示都依赖于原文本.而通过小波树,计数查询可以完全不依赖于原文本.因此,将后缀数组与小波树相结合可以实现不依赖于原文本的索引技术.

3.2.2 压缩的自索引

为了进一步提高后缀数组的空间效率,可以对后缀数组进行适当的压缩.基于后缀数组的自索引按照压缩方式,可分为基于定位函数压缩的自索引与基于 Lempel-Ziv 压缩[52]的自索引两种.

(1) 基于定位函数压缩的自索引是目前研究最多的自索引,其核心思想是:将小波树与后缀数组相结合,并

对后缀数组按照一定的间隔 b 进行后缀采样存储.由于小波树存储了编码后的二进制序列及相应的字符对应的编码信息,因此,可以实现位置 r 与 l 之间的文本显示.进而达到只需要小波树、文本采样的实体以及相应的附加数组,而不需要原始文本就能实现文本的定位查询与显示功能.该方法进一步可分为压缩定位函数与压缩定位函数的反函数两类:

① Ferragina 与 Manzini 在文献[44]中首次提出了基于定位函数的后缀数组压缩自索引 FMI(index of Ferragina and Manzini).FMI 首先将文本 $T_{1,n}$ 进行 BWT(Burrows-Wheeler transform)变换[45]得到 $T^{\mathbf{bwt}}$,然后对 $T^{\mathbf{bwt}}$ 采用前移(move-to-front)变换[53],接着采用 run-length 压缩[54],形成一种变长前缀码.FMI 最大的特点在于,以压缩的方式实现了不依赖于原文本的自索引,但其空间效率与时间效率都不够理想.

为了提高空间效率,Sadakane[55]提出了基于小波树的自索引 WTI(wavelet tree index).WTI 的基本思想是:首先将文本 $T_{1,n}$ 进行 BWT 变换得到 $T^{\mathbf{bwt}}$,然后对 $T^{\mathbf{bwt}}$ 的序列进行小波树变换.由于对于有 σ 个字母集合上的文本序列 $T_{1,n}$,当采用小波树变换时,只需要 $nH_0(T)+O(n\log_2\log_2 n/\log_\sigma n)$ 比特,而且对于 $T_{1,n}$ 中的某个字符,其排序与选择操作的时间效率为 $O(\log_2\sigma)$,因此,WTI 在空间效率上有了很大的提高,且具有与字母的无关性,但其时间效率仍有待于改善.

Makinen 等人对 WTI 进行了改进,提出了基于 run-length 编码的自索引 RMI(run-length FM-index)[54,56].其思想是首先将文本 $T_{1,n}$ 进行 BWT 变换得到 $T^{\mathbf{bwt}}$,然后对于 $T^{\mathbf{bwt}}$ 的序列进行小波树变换,再对小波树采用 run-length 压缩以提高空间效率.RMI 不仅具有 FMI 索引实现的低复杂性,同时与字符数大小 σ 无关.RMI 不仅具有与文本的 k 阶经验熵[57]成比例的空间消耗,同时具有 $O(m)$ 的搜索时间效率.Ferragina 等人也对 WTI 进行了改进,提出了友好字符(alphabet-friendly)自索引 AFI(alphabet-friendly index)[58,59].AFI 的思想是,将文本 $T_{1,n}$ 进行 BWT 变换得到 $T^{\mathbf{bwt}}$,然后将 $T^{\mathbf{bwt}}$ 根据文本内容及长度分成很多个子字符串 T^s,再对各子字符串分别进行压缩,进而提高空间效率.AFI 与 WTI 一样,具有相同的时间效率,但都提高了空间效率.

② Sadakane[60,61]提出了基于定位函数的反函数的前缀数组压缩自索引 CSA.对于文本序列 $T_{1,n}$,CSA 并不能直接访问 $A[i]$,而是提供了直接访问前缀 $T_{A[i],n}$ 的方法,并通过定位函数的反函数以及附加的数据结构实现文本 $T_{1,n}$ 与后缀数组 A 的表示.为了实现快速查找,CSA 采用一个比特矢量及二进制搜索算法,在最坏的情况下,其搜索时间效率为 $O(m\log_2 n)$,但是 CSA 的空间效率并不高.

为了提高空间效率,Grossi 等人[51,62]提出了改进的前缀数组压缩自索引 GCSA(compressed suffix array of Grossi).GCSA 基于以下方法:对于大小为 σ 的字符集上的文本 $T_{1,n}$,进行 BWT 变换得到文本 $T^{\mathbf{bwt}}$,将 $T^{\mathbf{bwt}}$ 根据文本内容及长度分成 σ^k 个子字符串 T^s,然后对各子字符串分别进行压缩,当采用适当的压缩方法时,其存储空间为 $nH_k+O(n\log_2\sigma)$ 比特.GCSA 将定位函数的反函数 Ψ 的值根据文本的内容分成不同的列表,并采用全局的比特矢量表示文本分割时是否出现的信息.

(2) 基于 Lempel-Ziv 压缩的自索引与基于定位函数压缩的自索引的实现方法完全不同:前者通过存储辅助的压缩数据结构来支持文本的快速搜索与定位.该自索引由 4 部分组成:文本 $T_{1,n}$;文本 $T_{1,n}$ 基于 BWT 变换的反向文本 T^R;文本 $T_{1,n}$ 的 Lempel-Ziv 数字树 lzTrie;一种用于存储分解位置及提供快速查询的数据结构[63,64].基于 Lempel-Ziv 压缩的自索引 FM(Ferragina and Manzini)[65]的最大优点在于,对于文本 $T_{1,n}$ 及搜索文本 $S_{1,m}$,如果 $S_{1,m}$ 在文本中出现的次数为 occ,则搜索与定位这 occ 个文本 $S_{1,m}$ 的时间为 $O(m+occ)$.这是目前的索引中,搜索与定位时间效率最高的索引.

3.2.3 后缀数组压缩自索引的比较

上述两种类型的后缀数组压缩自索引的时空性能比较见表 4.由于定位函数及其反函数可以分别实现反向搜索与正向搜索,因此基于定位函数压缩的自索引在实现文本搜索、定位及显示功能时采用了两种不同的技术路线.当采用不同的压缩方法时,这些自索引具有不同的时间效率与空间效率.如 FMI,作为第 1 个基于后缀数组的压缩自索引,其空间开销最大,但其搜索时间最快,达到了 $O(m)$.AFI 与 WTI 都对 FMI 的空间效率进行了提高,它们具有相同的时间效率,但是 AFI 的空间效率比 WTI 要高.而 WTI 的最大特点是与字母的无关性,即其性能不依赖于组成文本的字母集合.对于基于定位函数的反函数压缩的自索引,虽然在搜索策略中采用了正向搜索

的新策略,如 CSA,但其时空效率并不理想,尽管 GCSA 对 CSA 的空间性能进行了改进,但与 AFI 等基于定位函数的压缩自索引相比,仍然存在一定的差距.

基于 Lempel-Ziv 压缩的自索引与基于定位函数的压缩自索引相比,采用了一种完全不同的策略:以存储辅助的压缩数据结构的方式来实现文本的快速搜索与定位.FM 作为一种基于 Lempel-Ziv 压缩的自索引,其最大优点在于搜索与定位搜索文本时间为 $O(m+occ)$,这是目前搜索与定位时间效率最高的索引,但其空间效率为 $O(nH_k \log_2^{\gamma} n)+O(n\log_2\sigma\log_2^{\gamma} n)$,仍有待于进一步提高.

Table 4 Performance comparison of compressed suffix array self-indexes

表 4 后缀数组压缩自索引的比较

Compression method	Index	Time efficiency: Search + location + display	Space efficiency (bit)	Character
Location function	FMI[44]	$O(m)+O(\log_2^{1+\varepsilon} n)+O(l+\log_2^{1+\varepsilon} n)$	$5nH_k+O(n\log_2\sigma)$	Lowest space efficiency, highest search efficiency
	WTI[55]	$O(m(1+\log_2\sigma/\log_2\log_2 n))$ $+O(\log_2^{1+\varepsilon} n\log_2\sigma/\log_2\log_2 n)$ $+O((l+\log_2^{1+\varepsilon} n)\log_2\sigma/\log_2\log_2 n)$	$nH_0+O(n\log_2\sigma)$	Improved the space efficiency of FMI
	RMI[54,56]	$O(m(1+\log_2\sigma/\log_2\log_2 n))$ $+O(\log_2^{1+\varepsilon} n\log_2\sigma/\log_2\log_2 n)$ $+O((l+\log_2^{1+\varepsilon} n)\log_2\sigma/\log_2\log_2 n)$	$nH_k\log_2\sigma+2n+O(n\log_2\sigma)$	Improved the space efficiency of WTI
	AFI[58,59]	$O(m(1+\log_2\sigma/\log_2\log_2 n))$ $+O(\log_2^{1+\varepsilon} n\log_2\sigma/\log_2\log_2 n)$ $+O((l+\log_2^{1+\varepsilon} n)\log_2\sigma/\log_2\log_2 n)$	$nH_k+O(n\log_2\sigma)$	Improved the space efficiency of WTI
	CSA[60,61]	$O(m\log_2 n)+O(\log_2^{\varepsilon} n)+O(l+\log_2^{\varepsilon} n)$	$nH_0/\varepsilon+O(n\log_2\log_2\sigma)+\sigma\log_2\sigma$	Based on inverse function
	GCSA[51,62]	$O(m\log_2\sigma+\log_2^{2+\varepsilon} n)$ $+O(\log_2^{1+\varepsilon} n\log_2\sigma/\log_2\log_2 n)$ $+O(l/\log_\sigma n)+\log_2^{1+\varepsilon} n)$	$nH_k/\varepsilon+O(n\log_2\sigma)$	Based on inverse function, improved the space efficiency of CSA
Lempel-Ziv	FM[65]	$O(m(1+\log_2\sigma/\log_2\log_2 n))$ $+O(1)+O(l+\log_2^{1+\varepsilon} n)$	$O(nH_k\log_2^{\gamma} n)+O(n\log_2\sigma\log_2^{\gamma} n)$	Highest search and location efficiency

尽管这些自索引的形成方法各异,时空性能也不尽相同,但这些自索引具有以下相同点:一是与后缀数组索引相比,全文检索的时空效率都有一定的提高;二是在索引的创建上,步骤多且复杂;三是在时间与空间性能上,这些自索引都存在一定的折衷,还不能获得理想的时空性能:即对于文本 T,无法在获得 $nH_k(T)$的理想空间性能的同时获得 $O(m)$的理想搜索时间性能.

3.3 两种自索引的比较

基于倒排文件的自索引与基于后缀数组的自索引的比较见表 5.由于两种自索引所基于索引的结构不同,分别采用了不同的压缩技术.倒排文件采用间隙编码降低倒排文件的空间消耗,而且在压缩的倒排文件上支持快速的查找与定位,方法是通过插入忽略信息尽可能地减少解压压缩数据,提高索引的动态性;而后缀数组则通过压缩定位函数、定位函数的反函数或采用 Lempel-Ziv 压缩的方式,其出发点是减少后缀数组的存储,其方法是在适当地存储其他辅助数据的前提下,通过支持二进制的搜索,以实现搜索文本的快速定位与显示功能.

Table 5 Performance comparison of two kinds of self-indexes

表 5 两种自索引的比较

Item	Inverted file self-index	Suffix array self-index
Compression method	d-gap, Golomb	Location function, inverse function, Lempel-Ziv
Depend on original text	Yes	No
Complexity	Low	High
Time and space efficiency	Improved simultaneity	Improved simultaneity

尽管这两种方法都能同时提高时空效率,尤其是对于后缀数组其时空性能改善显著,但由于索引技术、压缩方式等因素所存在的差异,使得两种自索引实现的复杂性相差很大:基于倒排文件的自索引由于结构清晰,索引项之间相互独立,索引复杂性低,实现开销小;而基于后缀数组的自索引由于采用了结构紧凑的二进制存储方式,尤其是索引元素之间相互依赖、耦合紧密,在压缩方式上相当复杂,因此实现开销大.

另外,与倒排文件相比,后缀数组自索引在存储空间、索引创建效率上仍存在较大差距.尤其是索引更新维护上无法实现部分更新.而可扩展性、可伸缩性与查询功能上都还有待于提高.因此,尽管已有少量研究机构建立了基于后缀数组的开放实验平台[66],但还没有真正具有适用性的应用系统.而基于倒排文件的自索引技术却继承了倒排索引实现简单、支持排序查询、可扩展性与可伸缩性强、易于更新的诸多优势,在很多系统中得到了广泛应用.

但是基于后缀数组的自索引具有检索效率高的优势,其最大特点在于,它不依赖于原文本,只需要基于索引数据即可实现文本的搜索、定位及文本显示等功能.与此相比,基于倒排文件的自索引稍逊一筹:虽然文本的搜索与定位功能可由索引实现,但是文本显示仍然依赖于原文本.

4 总结与展望

索引技术、索引与压缩混合技术以及自索引技术分别从不同的角度出发提高全文索引的时间效率与空间效率,3 类技术的比较见表 6.

Table 6 Performance comparison of three kinds of index techniques

表 6 3 类索引技术的比较

Item	Index technique	Hybrid of index and compression technique	Self-Index technique
Time efficiency	Improved	Improved	Improved
Space efficiency	Degraded	Improved	Improved
Time and space contradiction	Yes	Yes	No
Complexity	Low	Medium	High

索引技术的前提是通过存储辅助数据,在适当地牺牲空间效率的前提下,显著地提高了全文检索的时间效率.作为一种简单而有效的技术,索引技术得到了极大的发展,并获得了广泛的应用.但是索引技术在时间与空间效率上还存在一定的对立性,签名文件的大小至少是原文本的 30%,倒排索引的大小是原文本的 50%~300%,而后缀树与后缀数组的存储空间消耗至少是原文本的 10 倍.

索引与压缩技术的结合,极大地提高了索引的空间效率.通过压缩编码技术,倒排文件的大小将降低到原文本大小的 10%,同时,内存空间、CPU 资源、I/O 操作、索引创建时间等性能都有了明显的改善;对于后缀数组,压缩后的空间效率则更加明显,从压缩前的至少为原始文本的 10 倍,减小到可能小于原文本的大小.但是由于数据压缩所带来的解码时间、编解码的复杂性在一定程度上降低了全文索引的时间效率.时间与空间效率的对立性仍然存在.尽管如此,索引与压缩技术相结合仍然是改进全文检索时空效率的有效方法.相比而言:压缩倒排索引仍然是应用最为广泛的索引,而且其应用面还在不断扩展,如支持结构化的 XML 查询、支持 P2P 的文本信息检索以及基于内容的多媒体音乐检索等新领域;而对后缀数组压缩技术的研究,由于其复杂性等因素具有相当的挑战性,尤其是在空间效率与可扩展性上还有很多工作需要深入;压缩索引在存储空间效率、查询时间效率等性能上还有待进一步地提高,尤其是如何对空间效率与时间效率的性能进行合理的折衷具有相当的难度.当前大多数的方法要么只针对其中的一个性能进行研究,要么提高了一个性能而另一个性能却出现恶化.选择合适的编码方式以及对索引的结构进行合理的改进,仍然是提高压缩索引空间效率与时间效率的主要途径.

自索引技术作为索引与压缩技术的进一步发展,使得时间效率与空间效率的同时提高成为可能,一定程度地解决了时间与空间效率的矛盾问题.例如,对于同一数据集,采用基于倒排文件的自索引技术与不采用自索引技术相比,对于 5~10 个词的连接布尔查询,处理时间减小约 5 倍;对于排序查询,处理时间减小约 2~4 倍.而插入忽略信息的大小约为原始文本大小的 2%.通过采用递归的方式对插入忽略信息的列表进行索引,并采用压缩技术,可以实现内存空间、存储空间与处理时间上的同时减小.而对于基于后缀数组的自索引,不仅不依赖于原始

文本,而且还能有效地实现任意子字符文本的重构.同时其空间效率与时间效率都得到了极大的改进,如基于定位函数压缩的自索引 FMI 的搜索时间消耗已达到 $O(m)$,基于 Lempel-Ziv 压缩的自索引 FM 的搜索与定位搜索文本时间达到了 $O(m+oc)$.

自索引技术作为一种提高全文检索时空效率的极具吸引力的新技术,由于其研究尚处于起步阶段,因而仍有很多问题需要进一步的研究与完善.

一是基于倒排文件的自索引研究.倒排索引作为一种应用广泛的索引技术,其时空性能还有待于进一步提高.基于倒排文件的自索引在文本显示时仍然依赖于原文本,从这个角度而言,文献[46]中提出的索引技术还不能说是真正意义上的自索引,而是一种采用间隙压缩方式的压缩索引.只有从倒排索引的存储方式上展开深入的研究,结合适当的压缩技术,并对索引的创建、维护与更新等方面进行根本的改进,才能从本质上实现时空效率的进一步提高.如由单级索引结构到多级索引结构、由静态倒排列表结构到动态列表结构、由单一压缩编码到混合压缩编码等.

二是基于后缀数组的自索引研究.目前压缩后缀数组索引的建立必须首先建立未压缩的后缀数组,这种方式不仅增加了空间开销,也增加了索引创建的时间开销.如何实现压缩索引的直接生成还有待于深入的研究.另外,后缀数组并不支持排序查询、近似匹配、压缩匹配、间隙匹配等扩展功能,而这方面的研究才刚刚起步,后缀数组的可扩展性还需进一步完善.后缀数组的动态更新也是一个问题,目前的研究大多集中在静态未压缩的后缀数组上,当有文本更新时,只能采用索引完全重建的方法实现索引的更新,这样不仅费时,而且影响了检索的准确性.如何实现后缀数组的动态更新与维护,也就成为未来研究值得关注的又一个方向.

致谢 在此,我们向对本文的工作给予支持和建议的审稿专家与同行,以及武汉大学软件工程国家重点实验室数据库新技术研究小组的老师和同学表示衷心的感谢.

References:

[1] The UC Berkeley report. How much information? Report 1, UC Berkeley, 2003. 1−30.

[2] Sievert MC. Full-Text information retrieval: Introduction. Journal of the American Society for Information Science, 1996,47(4):261−262.

[3] Liu XZ, Sun S, Zeng C, Peng ZY. An inverted index mechanisms based on buffers. Journal of Computer Research and Development, 2007,44(S):153−158 (in Chinese with English abstract).

[4] Fox E, Harman D, Baeza-Yates R, Lee W. Inverted files. In: Frakes B, Baeza-Yates RA, eds. Information Retrieval: Data Structures and Algorithms. Englewood Cliffs: Prentice-Hall, 1992.

[5] Zobel J, Moffat A. Inverted files for text search engines. ACM Computing Surveys, 2006,38(2):1−56.

[6] Carterette B, Can F. Comparing inverted files and signature files for searching a large lexicon. Information Processing and Management, 2005,41(8):613−633.

[7] Faloutsos C. Access methods for text. ACM Computing Surveys, 1985,17(1):49−74.

[8] Faloutsos C, Christodoulakis S. Signature files: An access method for documents and its analytical performance evaluation. ACM Trans. on Office Information Systems, 1984,2(4):267−288.

[9] Mccreight EM. A space-economical suffix tree construction algorithm. Journal of the ACM, 1976,23(2):262−272.

[10] Manber U, Myers E. Suffix arrays: A new method for on-line string searches. SIAM Journal on Computing, 1993,22(5):935−948.

[11] Gonnet GH, Baeza Y, Snider RA. New indices for text: PAT trees and PAT arrays. In: Frakes B, Baeza-Yates RA, eds. Information Retrieval: Data Structures and Algorithms. Englewood Cliffs: Prentice-Hall, 1992.

[12] Shieh W Y, Chung C P. A statistics-based approach to incrementally update inverted files. Information Processing and Management, 2005,41(5):275−288.

[13] Bell TC, Moffat A, Craig GN, Witten IH, Zobel J. Data compression in full-text retrieval systems. Journal of the American Society for Information Science, 1993,44(9):508−531.

[14] Faloutsos C. Signature files. In: Frakes B, Baeza-Yates RA, eds. Information Retrieval: Data Structures and Algorithms. Englewood Cliffs: Prentice-Hall, 1992.

[15] Sacks-Davis R, Kent A, Ramamohanarao K. Multikey access methods based on superimposed coding techniques. ACM Trans. on Database System, 1987,12(4):655−696.

[16] Sacks-Davis R, Kent A, Ramamohanarao K, Thom J, Zobel J. Atlas: A nested relational database system for text applications. IEEE Trans. on Knowledge and Data Engineering, 1995,7(3):454−470.

[17] Bookstein A, Klein ST. Using bitmaps for medium sized information retrieval systems. Information Processing and Management, 1990,26(5):525−533.

[18] Weiner P. Linear pattern matching algorithm. In: Pery NH, eds. Proc. of the 14th Annual IEEE Symp. on Switching and Automata Theory. IEEE Press, 1973. 1−11.

[19] Ukkonen E. On-Line construction of suffix trees. Algorithmica, 1995,14(3):249−260.

[20] Kurtz S. Reducing the space requirements of suffix trees. Software—Practice and Experience, 1999,29(13):1149−1171.

[21] Abouelhoda M, Kurtz S, Ohlebusch E. Replacing suffix trees with enhanced suffix arrays. Journal of Discrete Algorithms, 2004,2(1):53−86.

[22] Zobel J, Moffat A, Kotagiri R. Inverted files versus signature files for text indexing. ACM Trans. on Database Systems, 1998,23(4):453−490.

[23] Zhou SG, Hu YF, Guan JH. Adjacency matrix based full-text indexing models. Journal of Software, 2002,13(10):1933−1942 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/13/1933.htm

[24] Liu XW, Tao XP, Yu Y, Hu YF. A new full-text index model-subsequence array model. Journal of Software, 2002,13(1):150−158 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/13/150.htm

[25] Elias P. Universal codeword sets and representation of the integers. IEEE Trans. on Information Theory, 1975,21(2):194−203.

[26] Bentley JL, Yao A. An almost optimal algorithm for unbounded searching. Information Processing Letter, 1976,5(3):82−87.

[27] Golomb SW. Run-Length encodings. IEEE Trans. on Information Theory, 1966,12(3):399−401.

[28] Gallager R, Voorhis DV. Optimal source codes for geometrically distributed integer alphabets. IEEE Trans. on Information Theory, 1975,21(2):228−230.

[29] Croft WB, Savino P, Dor O. Implementing ranking strategies using text signatures. ACM Trans. on Office Information Systems, 1988,6(1):42−62.

[30] Whitte IH, Moffat A, Bell TC. Compression and full-text indexing for digital libraries. SIGOIS Bulletin, 1994,15(1):11−13.

[31] Moffat A, Turpin A, Katajainen J. Space-Efficient construction of optimal prefix codes. In: Hec CC, ed. Proc. of the 5th IEEE Data Compression Conf. Snowbird: IEEE Computer Society Press, 1995. 192−201.

[32] Bookstein A, Klein ST, Raita T. Simple Bayesian model for bitmap compression. Information Retrieval, 2000,1(4):315−328.

[33] Witten IH, Moffat A, Bell TC. Managing Gigabytes: Compressing and Indexing Documents and Images. 2nd ed., San Francisco: Morgan Kaufmann Publishers, 1999. 50−73.

[34] Moffat A, Zobel J. Parameterised compression for sparse bitmaps. In: Seyit K, ed. Proc. of the ACM Int'l Conf. on Research and Development in Information Retrieval. New York: ACM Press, 1992. 274−285.

[35] Chiuen T, Varadarajan S. SASE: Implementation of a compressed text search engine. In: Mary CH, ed. Proc. of the 1st USENIX Symp. on Internet Technologies and Systems. USENIX Association, 1997. 47−56.

[36] Navarro G, Moura ESD, Neubert M. Adding compression to block addressing inverted indexes. Information Retrieval, 2000,3(1):49−77.

[37] Ciaccia P, Tiberio P, Zezula P. Declustering of key-based partitioned signature files. ACM Trans. on Database System, 1996,21(3):295−338.

[38] Kocberbera S, Can F. Vertical framing of superimposed signature files using partial evaluation of queries. Information Processing and Management, 1997,33(3):353−376.

[39] Lee DL, Kim YM, Patel G. Efficient signature file methods for text retrieval. IEEE Trans. on Knowledge and Data Engineering, 1995,7(3):423−435.

[40] Faloutsos C. Signature files: Design and performance comparison of some signature extraction methods. In: Ciaccia P, ed. Proc. of the 8th ACM-SIGIR Conf. on Information Retrieval. New York: ACM Press, 1985. 63−82.

[41] Ciaccia P, Zfzula P. Estimating accesses in partitioned signature file organizations. ACM Trans. on Information System, 1993,11(2):133−142.

[42] Kazutaka F, Kazushige A, Atsushi I. Implementation and performance evaluation of compressed bit-sliced signature files. In: Can F, ed. Proc. of the Conf. on Information Systems and Management of Data. New York: ACM Press, 1999. 164−177.

[43] Seyit K, Fazh C. Compressed muti-framed signature files: An index structure for fast information retrieval. In: Kurtz S, ed. Proc. of the System Algorithm Conf. New York: ACM Press, 1999. 221−226.

[44] Ferragina P, Manzini G. Opportunistic data structures with applications. In: Atsushi I, ed. Proc. of the 41st IEEE Symp. on Foundations of Computer Science (FOCS). New Jersey: IEEE Press, 2000. 390−398.

[45] Burrows M, Wheeler D. A block sorting lossless data compression algorithm. Technical Report, 124, Digital Equipment Corporation, 1994. 1−24.

[46] Moffat A, Zobel J. Self-Indexing inverted files for fast text retrieval. ACM Trans. on Information Systems, 1996,14(4):349−379.

[47] Makinen V. Compact suffix array—A space-efficient full-text index. Fundamenta Informaticae, 2003,56(1-2):191−210.

[48] Grossi R, Vitter JS. Compressed suffix arrays and suffix trees with applications to text indexing and string matching. SIAM Journal on Computing, 2005,35(2):378−407.

[49] Couvreur TR, Benzel RN, Miller SF, Zeitler DN, Lee D L. An analysis of performance and cost factors in searching large text databases using parallel search systems. Journal of the American Society for Information Science, 1994,45(7):443−464.

[50] Gonzalo N, Veli M. Compressed full-text indexes. ACM Computing Surveys, 2007,39(1):1−61.

[51] Grossi R, Gupta A, Vitter J. High-Order entropy-compressed text indexes. In: Rauhe T, ed. Proc. of the 14th Annual ACM-SIAM Symp. on Discrete Algorithms (SODA). New York: ACM Press, 2003. 841−850.

[52] Lempel A, Ziv J. On the complexity of finite sequences. IEEE Trans. on Information Theory, 1976,22(1):75−81.

[53] Bentley J, Sleator D, Tarjan R, Wei V. A locally adaptive compression scheme. Communications of the ACM, 1986,29(4):320−330.

[54] Makinen V, Navarro G. Succinct suffix arrays based on run-length encoding. Lecture Notes in Computer Science, 2005,3537: 45−56.

[55] Sadakane K. Succinct representations of LCP information and improvements in the compressed suffix arrays. In: Tarjan R, ed. Proc. of the 13th Annual ACM-SIAM Symp. on Discrete Algorithms (SODA). New York: ACM Press, 2002. 225−232.

[56] Makinen V, Navarro G. Run-Length FM-index. In: Rauhe T, ed. Proc. of the Burrows-Wheeler Transform: Ten Years Later. New York: ACM Press, 2004. 17−19.

[57] Manzini G. An analysis of the Burrows-Wheeler transform. Journal of the ACM, 2001,48(3):407−430.

[58] Ferragina P, Manzini G, Makinen V, Navarro G. An alphabet-friendly FM-index. In: Apostolico A, ed. Proc. of the 11th Int'l Symp. on String Processing and Information Retrieval (SPIRE). Berlin, Heidelberg: Springer-Verlag, 2004. 150−160.

[59] Ferragina P, Manzini G, Makinen V, Navarro G. Compressed representation of sequences and full-text indexes. ACM Trans. on Algorithms, 2007,3(2):1−24.

[60] Sadakane K. Compressed text databases with efficient query algorithms based on the compressed suffix array. In: Apostolico A, ed. Proc. of the 11th Int'l Symp. on Algorithms and Computation (ISAAC). LNCS 1969, Berlin, Heidelberg: Springer-Verlag, 2000. 410−421.

[61] Sadakane K. New text indexing functionalities of the compressed suffix arrays. Journal of the Algorithms, 2003,48(2):294−313.

[62] Grossi R, Gupta A, Vitter J. When indexing equals compression: Experiments with compressing suffix arrays and applications. ACM Trans. on Algorithms, 2006,2(4):611−639.

[63] Karkkainen J, Ukkonen E. Lempel-Ziv parsing and sublinear-size index structures for string matching. In: Gupta A, ed. Proc. of the 3rd South American Workshop on String Processing (WSP). Carleton University Press, 1996. 141−155.

[64] Alstrup S, Brodal G, Rauhe T. New data structures for orthogonal range searching. In: Vitter J, ed. Proc. of the 41st IEEE Symp. on Foundations of Computer Science (FOCS). IEEE Press, 2000. 198−207.

[65] Ferragina P, Manzini G. Indexing compressed texts. Journal of the ACM, 2005,52(4):552−581.

[66] PizzaChili project. 2005. http://pizzachili.dcc.uchile.cl

附中文参考文献:

[3] 刘小珠,孙莎,曾承,彭智勇.基于缓存的倒排索引机制研究.计算机研究与发展,2007,44(S):153–158.

[23] 周水庚,胡运发,关佶红.基于邻接矩阵的全文索引模型,软件学报,2002,13(10):1933–1942. http://www.jos.org.cn/1000-9825/13/1933.htm

[24] 刘学文,陶晓鹏,于玉,胡运发.一种全新的全文索引模型—后继数组模型,软件学报,2002,13(1):150–158. http://www.jos.org.cn/1000-9825/13/150.htm

刘小珠(1977－),女,湖北潜江人,博士生,讲师,主要研究领域为全文索引技术,基于对等网络的全文检索技术.

彭智勇(1963－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为复杂数据管理,可信数据管理,Web 数据管理.