ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.18, No.4, April 2007, pp.919–932
DOI: 10.1360/jos180919

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

XML 信息检索中最小子树根节点问题的分层算法*

孔令波[1+]，唐世渭[1,2]，杨冬青[1]，王腾蛟[1]，高 军[1]

[1](北京大学 计算机科学技术系,北京 100871)

[2](北京大学 视觉与听觉信息处理国家重点实验室,北京 100871)

Layered Solution for SLCA Problem in XML Information Retrieval

KONG Ling-Bo[1+], TANG Shi-Wei[1,2], YANG Dong-Qing[1], WANG Teng-Jiao[1], GAO Jun[1]

[1](Department of Computer Science and Technology, Peking University, Beijing 100871, China)

[2](National Laboratory on Machine Perception, Peking University, Beijing 100871, China)

+ Corresponding author: Phn: +86-10-62755440, E-mail: lbkong@db.pku.edu.cn

Kong LB, Tang SW, Yang DQ, Wang TJ, Gao J. Layered solution for SLCA problem in XML information retrieval. *Journal of Software*, 2007,18(4):919–932. http://www.jos.org.cn/1000-9825/18/919.htm

Abstract: SLCA (smallest lowest common ancestor) problem is a basic task of keyword search in XML information retrieval. It means to find all the nodes corresponding to the tightest subtrees in XML data, which involves the given keywords. Xu, *et al*., illustrate three algorithms—Indexed lookup eager (ILE), stack algorithm and scan eager (SE), and manifest that ILE has the best performance. Different from the complicated-B+-tree-based ILE algorithm, this paper proposes a layered solution for SLCA problem, named as LISA (layered intersection scan algorithm). It benefits from the distribution rule of SLCA nodes in XML tree, and calculates the SLCA nodes level by level (the deepest level runs first). That is, based on the retrieved Dewey codes corresponding to given keywords, the Dewey codes of SLCA nodes can be gotten by intersecting the prefix Dewey codes of each level. Compared with the ILE algorithm, LISA solutions need not sophisticated data structures, and have comparatively runtime performance. There are two instances following the LISA idea, called LISA I and LISA II respectively. They are distinguished from each other according to whether keeping Dewey codes in computation or transforming Dewey codes into integer sequences. Extensive experiments evaluate the performance of algorithms and prove the efficiency of LISA II.

Key words: XML index; Dewey code; XML information retrieval; keyword search; SLCA (smallest lowest common ancestor); ILE (indexed lookup eager)

摘 要: 最小子树根节点问题(smallest lowest common ancestor,简称 SLCA)是实现 XML 信息检索研究中关键字查询的一个基本问题,其主旨就是求解所有包含给定关键字的紧致子树的根节点.XU 等人给出了 3 种算法—基于索引的搜索算法(indexed lookup eager,简称 ILE)、基于堆栈的算法以及基于扫描的算法(scan eager,简称 SE),并通过实验证明 ILE 算法具有最好的表现.与基于 B+树索引结构的 ILE 算法不同,所给出的新算法,称为 LISA(layered intersection scan algorithm)方法.该方法基于 SLCA 节点按“层”分布的规律,采取了逐层求解 SLCA 节点的思路,即在

* Supported by the National Natural Science Foundation of China under Grant No.60503037 (国家自然科学基金); the National High-Tech Research and Development Plan of China under Grant No.2005AA4Z307 (国家高技术研究发展计划(863)); the Beijing Natural Science Foundation of China under Grant No.4062018 (北京市自然科学基金)

Received 2005-10-20; Accepted 2006-04-27

获取了包含关键字的节点的 Dewey 码集合后,通过计算对应于不同关键字、不同层次的 Dewey 码前缀集合的交集,可以得到对应不同层的 SLCA 节点.与 ILE 相比,LISA 除了只需对应于关键字的节点集合信息以外,不再需要其他复杂的辅助数据结构——全部的信息只是对应不同关键字的 Dewey 码集合以及排序操作.同时,给出了两种实际的算法:LISA I 和 LISA II,二者的区别在于是否采用 Dewey 编码到整数的转换.其中,LISA II 更具有满意的性能.

关键词: XML 索引;Dewey 编码;XML 信息检索;关键字查询;SLCA;ILE
中图法分类号: TP311 文献标识码: A

XML 数据的广泛应用,使得从 XML 数据中获取有用的信息成为当前数据库领域研究的热点[1].主流的解决思路与关系数据库方案的方式类似:首先定义格式良好的查询语言,之后使用者必须学会如何使用这种语言定义他们的查询要求,实际的系统根据用户的查询模式与储存的数据进行匹配,得到满足查询模式的结果.可见,这里同样存在对普通用户使用门槛过高的问题.为此,研究人员尝试将对用户友好的信息检索处理的思路引入 XML 数据处理研究.在这一过程中,将关键字查询直接用于 XML 数据的思路,以其最大保留了对普通用户的便利而得到关注.与普通文档上的关键字查询不同,XML 数据上关键字查询的目标通常不是整个 XML 文档,而是满足给定关键字的最紧致 XML 片段.文献[2]将该问题归结为 SLCA(smallest lowest common ancestor)问题,是 XML 关键字查询的基本任务.

基于 XML 数据的 Dewey 编码[3,4],文献[2]给出了 3 种算法:Indexed Lookup Eager(ILE)算法、Scan Eager(SE)算法和 Stack 算法,并通过实验证明 ILE 算法具有最好的性能.ILE 算法的基本思路是:将 XML 数据保存到 B+树结构,插入 B+树的数据形式为(keyword,dewey),这相当于将 XML 中的数据按照关键字(keyword)和关键字在 XML 树中的节点 Dewey 码进行排序;之后,借助于 B+树结构以及 Dewey 码的基本运算(大于、小于、子孙码、最长公共前缀等)计算 SLCA 节点.ILE 算法存在如下不足:一是必须修改 B+树结构支持必要的 Dewey 码操作,其实现较为复杂.而且,虽然 B+树索引结构在现有的数据库管理系统中普遍使用,但是都不适用于 Dewey 编码数据.因此,如果试图考虑在实际应用中利用已有成熟的关系数据库管理系统,则代价较大;二是 ILE 算法计算 SLCA 的过程是“粗暴(brute)”的,即首先将全部 XML 数据保存入改造过的 B+树;之后,根据获取对应不同关键字的 Dewey 码集合;然后,反复调用 Dewey 码的基本操作和 B+树上的匹配操作来求解 SLCA 节点.相对于只依赖对应不同关键字的 Dewey 码集合以及简单的辅助数据结构而言,ILE 算法显得更为复杂;而且,根据我们实际实验的情况,ILE 算法的实际性能也缺乏一定的优势.更详细的描述参见第 1.2 节内容和第 3 节的实验结果.

针对前述 ILE 的不足,本文算法的设计基于如下考虑:新算法应较少依赖复杂辅助数据结构,而是主要依赖于对应不同关键字获得的 Dewey 码集合;算法的运行性能有良好的表现.为此,本文仔细考虑了 SLCA 节点必然按“层”分布这一特点,尝试着将该特点纳入 SLCA 问题的求解.基本的思路是:将对应不同关键字的 Dewey 码按照层次的不同分解为相应的前缀;之后,从最深一层开始逐层求解对应层上的 SLCA 节点.在每一层求解 SLCA 节点的问题就转换为求解对应不同关键字的 Dewey 码集合的交集运算.由于对应某紧致 XML 片段的 SLCA 节点必然唯一地分布于某一层,所以,根据上述思路必然能够求得全部的 SLCA 节点.同时,由于采取了逐层求解的方式,因此,对应 i 层 SLCA 节点的 Dewey 码就不必参与(i–1)层的 SLCA 节点的求解,从而降低了计算的冗余.本文将秉承这一逐层求解 SLCA 节点思路的算法统一称为 LISA(layered intersection scan algorithm)算法,并根据是否将 Dewey 码转换为整数参与计算的不同而设计了两种实际的 LISA 实例,分别称为 LISA I 和 LISA II,后者利用了 Dewey 码到整数的转换.实验结果表明,LISA II 算法有着较高的性能.

本文的主要工作概括如下:

(1) 提出了采取分层求解 SLCA 节点的 LISA 思路,并设计和实现了两种实际的算法;

(2) 探讨了新思路的可行性,并对 LISA II 算法作了计算复杂度分析;

(3) 实现相关的 SLCA 求解算法,并进行了大量的实验,比较和分析了实验的结果.

本文第 1 节给出研究的背景以及相关工作,重点介绍文献[2]中的 ILE 算法,深入分析该算法存在的不足.第 2 节首先给出分层求解 SLCA 问题的 LISA 算法的思路,之后结合实际的例子介绍基于 LISA 思路的两个实

现——LISA I 和 LISA II,并分析 LISA II 算法的复杂性.第 3 节给出实验结果.最后是全文的总结和研究展望.

1 问题描述和相关工作

1.1 问题描述

XML 规范已成为当前网络应用中数据表达和交换的事实上的标准,满足 XML 规范的 XML 数据广泛应用于电子商务、数字图书馆和 Web 服务中.为便于 XML 数据的处理,研究中通常将 XML 数据映射为树状结构,称作 XML 树[5].任一节点 u 都有一条从根节点到该节点 u 的路径存在,路径经过的所有节点称为节点 u 的路径节点;路径节点上标签的序列称为节点 u 的标签路径.标签路径的概念是基于正则路径查询语言处理 XML 数据的基础.这是因为:如果给定标签序列,在遍历 XML 树的过程中,通过比较给定标签路径与 XML 树中节点的标签路径是否匹配,即可得到满足给定标签路径的节点的集合.

为了从 XML 以及半结构化数据中获取所需要的信息,研究人员开发了许多查询语言,包括 Lorel,XML-QL,XML-GL,Quilt,XPath,XQuery.它们共同的特征为:采用正则路径表达式的形式,其本质是捕捉 XML 数据单元间的结构关系和内容.为了提高 XML 查询处理的效率,如何巧妙设计能够捕捉节点间结构关系,并便于结构关系计算的 XML 索引,成为 XML 查询处理研究的热点[4].

但是,这种 XML 数据处理方式具有类似数据库查询方式的缺陷:只有用户掌握了复杂的查询语言的语法并且了解数据的组织结构后,才能借助查询语言描述自己的查询请求,因而不利于普通用户的使用.为此,将信息检索(information retrieval,简称 IR)的数据处理方式纳入 XML 的数据处理(XML IR),就成为当前 XML 数据研究的新的思路.其中,作为最贴近普通用户的关键字查询形式,在 XML 信息检索处理中得到了最先的关注.如何得到贴合关键字集合的最紧致的 XML 片段,即文献[2]中的 SLCA 问题,就成为这类研究的基本点.定义如下:

定义 **1**(**SLCA** 问题). 在给定如下初始条件的情况下:一是对应 XML 数据的标签有向树 $G=(V_G,E_G,r,A)$,其中:V_G 表示树中节点的集合;E_G 表示树中所有边的集合;r 为标签有向树的根;A 表示所有节点标签的集合;二是关键字序列 $W=\{w_1,w_2,\ldots,w_k\}$.SLCA 问题就是求解 G 中所有满足如下条件的子树的根节点:

(1) 子树必须包含关键字序列 W,即 W 中的任一关键字必然分布于该子树的叶节点;

(2) 子树中不存在更小的子树同样包含 W.

这样的子树也称为满足关键字序列 W 的最紧致 XML 片段.

图 1 是基于一个实际 XML 数据的 XML 树的实例.

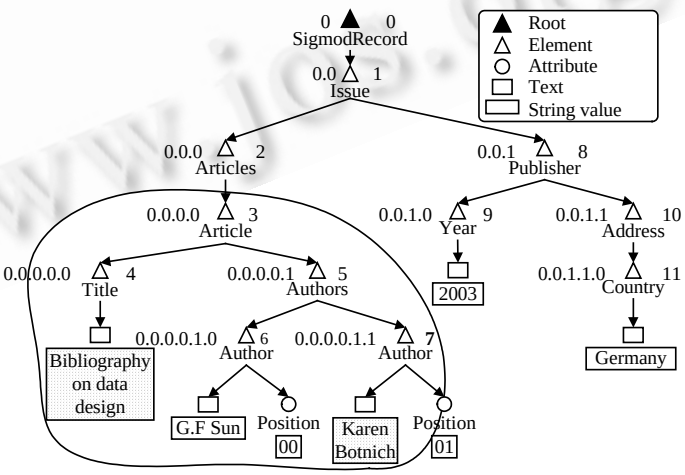

Fig.1 XML tree with Dewey codes and preNum codes

图 1 带有 Dewey 编码和 preNum 编码的 XML 树

图 1 中,元素节点左侧的整数序列就是该节点的 Dewey 编码,节点右侧的整数为该节点出现在前序遍历

XML 树的序列中的位置编号.其中:两个阴影框表示关键字“Bibliography Botnich”所在的节点,环内框住的子树即为满足给定关键字的最紧致 XML 片段,子树的根节点“article”即为 SLCA 节点.

针对 SLCA 问题的通行的解法可概括如下:首先获取分别包含对应关键字的叶节点的集合,标记为 $S_i(1\le i\le k)$,即 S_i 为包含关键字 w_i 的所有叶节点的集合;之后,根据节点的信息(例如,节点的编码)求解 SLCA 节点.为了实现这一过程,赋予节点的编码信息应当具有重构 XML 片段的能力,这就要求能够根据编码得到其路径上其他节点信息的能力.考察现有的 XML 索引技术[4],局部编码具有此能力.其中,Dewey 编码以其简单、易理解而成为 XML 关键字查询中解决 SLCA 问题的基本技术.

1.2 相关工作

相关工作包括两个方面:一是 Dewey 编码及其基本操作;二是 ILE 算法处理 SLCA 问题的讨论.

1.2.1 Dewey 编码及其基本操作

根据文献[4]的叙述,局部编码具有包含路径上其他节点信息的能力.其中,Dewey 编码以其简单、易理解而成为求解 SLCA 问题的基本技术.定义如下:

定义 **2**(**Dewey** 编码). 给定对应 XML 数据的标签有向树 $G=(V_G,E_G,r,A)$,G 中任意节点的 Dewey 编码由下列规则确定:

(1) 根节点 r 的 Dewey 编码为‘0’;

(2) 在宽度优先遍历 G 的过程中,如果节点 v 是节点 u 的第 i 个孩子节点,那么,节点 v 的 Dewey 码为‘$D(u).i-1$’.其中的 $D(u)$表示节点 u 的 Dewey 码.

Dewey 码 u 中所有被‘.’分割的整数的个数表示该 Dewey 码 u 的长度,以 l_u 表示.取 XML 树中根节点 r 所在的层为 1,那么称 Dewey 码 u 中与第 i 层节点对应的整数为该 Dewey 码的第 i 层整数,表示为 $u(i)$.由 1 到 i 层整数组成的 Dewey 码称为该节点 Dewey 码的第 i 层前缀,表示为 $p_u(i)$.Dewey 码的基本运算列举如下:

设 u 和 v 表示两个 Dewey 码,长度分别为 l_u 和 l_v.

性质 **1**(相等关系). 如果 $l_u=l_v$,并且从左至右逐层比较各层的整数都相等,则称 v 等于 u,表示为 $u=v$.

性质 **2**(包含关系). 如果 $l_u<l_v$,并且 $p_v(l_u)=u$,则称 v 包含 u.

性质 **3**(大小判断).

(1) 如果 v 包含 u,则称 v 大于 u,表示为 $v>u$;

(2) 从左至右顺序比较 u,v 各层的整数,仅当第 i 层时二者的整数不同,如果 $u(i)<v(i)$,则称 v 大于 u.

性质 **4**(后代 **Dewey** 码). 表示为 $descendent(u,v)$.

(1) 如果参数 u,v 其中之一为空,则返回另一个非空的 Dewey 码;

(2) 如果 v 包含 u,则返回 v.

性质 **5**(公共前缀). 表示为 $lca(u,v)$,计算如下:

(1) 如果二者具有包含关系,那么返回被包含的 Dewey 码;

(2) 从左至右顺序比较 u,v 各层的整数,仅当第 i 层时二者的整数不同,那么,$p_u(i-1)$即为二者的公共前缀.

根据 Dewey 码的定义,$p_u(i)$对应 u 节点路径上第 i 层节点的 Dewey 码.可见,给定一个 Dewey 码也就自然地得到了该节点路径上所有节点的 Dewey 码.各运算的时间复杂度可统一归结为 $O(l_{lca(u,v)})$.

1.2.2 基于 Dewey 码求解 SLCA 问题

针对 SLCA 问题,文献[2]给出了 3 种基于 Dewey 编码计算 SLCA 的算法:Indexed Lookup Eager(ILE)算法、Scan Eager (SE)算法和 Stack 算法,文中的实验数据表明,ILE 算法具有较优的性能表现.本文主要针对 ILE 算法进行对比分析.

ILE 的基本思路可归结为如下步骤:

(1) 设计支持(关键字,Dewey 码)数据格式的 B+树结构(Dewey B-plus tree,简称 DBPT),并预先将 XML 数据分解保存到该数据结构中.新的 B+树结构应实现如下两个操作:$lm(dewey,keyword)$ 和 $rm(dewey,keyword)$,分别表示获取关键字为“keyword”的 Dewey 码集合中小于/大于给定 Dewey 码

“dewey”的最大/最小 Dewey 码.

(2) 获取对应给定关键字序列的 Dewey 码集合.每个关键字对应一个包含该关键字的节点的 Dewey 码集合,并将全部集合按照集合内元素的多少从小到大排序,用 $D_1,D_2,\ldots,D_k$ 表示,其中 S_1 对应元素数目最小的 Dewey 码集合.

(3) 根据文献[2],从 $D_1,D_2,\ldots,D_k$ 中求解 SLCA 过程可通过如下公式说明:

$$slca(D_1,...,D_k)=removeAncestor\left(\bigcup_{v\in D_1} slca(\{v\},D_2,...,D_k)\right) \tag{1}$$

其中,$slca(\{v\},D_1,\ldots,D_k)$的计算公式如下:

$$slca(\{v\},D_1,\ldots,D_k)=slca(slca(\{v\},D_1,\ldots,D_{k-1}),D_k) \tag{2}$$

而从某集合中求取对应单个 Dewey 码的 SLCA 节点的过程如下公式所示

$$slca(\{v\},D)=\{descendant(lca(v,lm(v,D)),lca(v,rm(v,D)))\} \tag{3}$$

通过上面的描述可以看出,ILE 算法存在如下不足之处:

(1) 现有数据库管理系统中虽然普遍存在 B+树结构的索引形式,但要支持 Dewey 码以及 *lm*,*rm* 操作,修改较为复杂.如果试图将之纳入成熟的关系数据库系统——这是当前数据库研究领域的一个方向,就必须深入系统内部进行修改,代价过高.

(2) 通过考察整个 ILE 过程可以看出,创建的 Dewey 码 B+树是基于整个 XML 数据的.那么,从某个集合中求解与单个 Dewey 码相对应的 SLCA 节点的基本计算,即公式(3),就是在整个 XML 数据构建的 Dewey 码 B+树上执行 *lm*,*rm* 的过程.通常,XML 数据本身的节点数目会远远大于包含关键字的节点数目,那么,在基于 XML 数据本身的节点构建的 B+树上执行 *lm*,*rm* 运算,与单纯的基于获得的 Dewey 码集合上的运算相比,就会存在大量的冗余,性能也会降低.

还需要指出的是,Dewey 码 B+树本身的构建也具有较高的时间代价.在内存中构建 Dewey 码 B+树的实验也证明了这一点.

(3) 正是由于公式(2)不能保证所得的临时节点就是最后的 SLCA 节点,所以,ILE 中还需有公式(1)的过滤过程,因而增加了整个算法的计算代价.

上述不足促使我们重新考虑 SLCA 问题的求解,并确定新算法应具有如下特点:尽可能仅利用获得的对应于关键字的 Dewey 码集合高效地求取 SLCA 节点.如此执行,SLCA 算法与成熟的数据库管理系统的结合就不成问题.

1.2.3 一点说明

为了能对 SLCA 问题有更深入的认识,对 XML 关键字查询在 XML 数据查询中的地位有所了解是有帮助的,所以,在此单独辟一小节作一简单的概括.

如果说类似 Xpath,XQuery 的 XML 查询处理,即首先精心定义满足 XML 内容和结构要求的查询语言,之后搜索 XML 数据、匹配满足使用查询语言描述的模式从而得到最后的结果是近期 XML 数据处理的主要研究内容,那么,将传统信息检索中的优点纳入到 XML 数据的处理,则是希望提供给用户一种友好的使用形式,成为当下 XML 数据处理新的研究点之一.近期关于这方面的研究可分为两部分:一是扩展前述面向模式匹配的查询语言(如 XPath,XQuery)以支持部分 XML 信息检索中的特点[6−11];另一部分就是将对用户使用友好的关键字查询延伸至 XML 数据[12−16].

由于第 1 种结合不能摆脱原有查询语言对普通用户的门槛要求——用户必须知晓 XML 数据的结构模式以及掌握复杂的语法,所以,本质上并不能降低对普通用户的限制,不符合 XML 信息检索的要求.所以,将关键字查询应用到 XML 数据就成为值得深入探讨的问题.不同于普通的文本文档,XML 文档是具有半结构、自描述特点的数据,针对这类数据的关键字查询具有了新的特点:查询结果通常不是整个文档,而是包含给定关键字的更小粒度的 XML 片段;结构相似性也成为筛选查询结果的重要要求.前者是后者的基础.文献[2]中的 SLCA 问题的讨论正是基于这一背景而提出来的.

2 SLCA 问题的 LISA 求解

2.1 LISA的思路

通过观察,SLCA 节点具有如下几个分布特征:(1) 如果某节点是给定关键字的 SLCA 节点,它必然唯一地属于某一“层”;(2) 根据 SLCA 的定义,如果 k 个分别包含给定的 k 个关键字的叶节点在 i 层有 SLCA 节点,则它们中的任何一个都不可能成为(i–1)层的 SLCA 节点所在子树的叶节点.

借助这两个事实,可以得到如下结论:已知 $d_1,d_2,\ldots,d_k$ 是该 SLCA 节点所在子树的包含关键字的 k 个叶节点的 Dewey 码,将它们分解为前缀 Dewey 码序列,并按层排列前缀,如果按照从深到浅的次序逐层观察来自不同关键字的前缀 Dewey 码,那么,对应各关键字的前缀 Dewey 码的公共元素必然是 SLCA 节点的 Dewey 码.这一点在文献[2]中阐述得已很清楚,之所以在此进一步叙述,是想说明如下的规律:在得到对应不同关键字的 Dewey 码集合($D_1,D_2,\ldots,D_k$,其中,D_i 表示包含关键字 w_i 的 Dewey 码集合)后,设 $\max L(D_i)$表示集合 D_i 中最长的 Dewey 码的长度,$\min\max L(D_1,D_2,\ldots,D_k)$表示所有 $\max L(D_i)$中最小的值,那么,可能的 SLCA 节点的 Dewey 码的长度必然属于$(2,\min\max L(D_1,D_2,\ldots,D_k)-1)$.由于第 1 层一定会对应根节点,它对 SLCA 问题而言没有实际的意义,所以最小为第 2 层.

进而,在 Dewey 码集合上求解 SLCA 节点的问题可以通过逐层求解的方式得到解决.首先从最深层$(\min\max L(D_1,D_2,\ldots,D_k)-1)$开始,获取 D_i 集合中对应该层的所有 Dewey 码的前缀,得到 D_1', D_2',…, D_k',各 D_i' 集合的交集如果不为空,则其中的元素就是第一批 SLCA 节点;在进一步求解对应较浅层的 SLCA 节点时,以前一层所得 SLCA 节点为前缀的所有 Dewey 码都应排除出去,即调整 $D_1,D_2,\ldots,D_k$ 中的数据;之后的 SLCA 计算与最深层上的计算类似——获取前缀码、计算它们的交集.当调整后的 $D_1,D_2,\ldots,D_k$ 中任一集合为空,或者已经到达第 1 层时,计算终止.

以上就是 LISA 算法的思路.从中可以看出,多个集合上的交集运算是 LISA 思路的核心问题.为了求解多个集合的交集,往往需要预先将集合按照元素的大小进行排序,交集的运算便转换为有序集合上的交集运算.如果参与计算的所有前缀都保留为 Dewey 码的形式,则称这种算法为 LISA I 算法;如果在计算之前将前缀统一映射为整数后再参与计算,则称这种算法为 LISA II 算法.

下面借助实例分别介绍两种算法的计算过程.查询的关键字组为“Bibliography Botnich”,对应关键字“Bibliography”的 Dewey 码集合 D_2 有 3 个元素——“0.0.1.0”(d_{21}),“0.0.0.0.0”(d_{22})和“0.1.0.0.0.0.0”(d_{23});对应“Botnich”的 Dewey 码集合 D_1 有 2 个元素——“0.0.0.0.1.1”(d_{11})和“0.0.1.1.0”(d_{12}).由数据可知,$\min\max L(D_1,D_2)=6$.SLCA 问题就转换为从这两个 Dewey 码集合中求解所有 SLCA 节点的 Dewey 码的过程.

2.1.1 LISA I 算法处理示例

将前述的数据用图 2 来表示,Level 一行的数字表示层的信息;两个方框分别包含了 D_1 和 D_2 的 Dewey 码数据,每个 Dewey 码各层上的整数由小方框包含,其位置与层号相对应.

LISA I 的处理首先从收集第 5 层的前缀开始,有 D_1' ={“0.0.0.0.1”}, D_2' ={“0.1.0.0.0”}.此时, D_1' 和 D_2' 的交集为空.LISA I 进入第 4 层的处理,此时的 D_1' ={“0.0.0.0”,“0.0.1.1”}, D_2' ={“0.1.0.0”,“0.0.0.0”},它们的交集为{“0.0.0.0”},其元素即为第 4 层上的 SLCA 节点 Dewey 码,在图 2 中由较深的灰色小框表示.

在进入第 3 层的处理之前,从 D_1 和 D_2 中删除以“0.0.0.0”为前缀的 Dewey 码,此时的 $D_1=\{d_{12}\}$,$D_2=\{d_{21},d_{23}\}$.那么,在第 3 层上求解 SLCA 时的 D_1' ={“0.0.1”},而 D_2' ={“0.0.1”,“0.1.0”},二者的交集为{“0.0.1”}.于是,我们得到第 3 层上的 SLCA 节点的 Dewey 码,在图 2 中由较浅的灰色小框表示.

与第 3 层的处理类似,当进入第 2 层时,首先从第 3 层的 D_1 和 D_2 中删除所有以“0.0.1”为前缀的 Dewey 码,此时的 $D_1=\{\}$,$D_2=\{d_{23}\}$.由于 D_1 已经为空,则整个 LISA I 算法终止.全部的 SLCA 节点的 Dewey 码就是从第 4、第 3 两层求得的 Dewey 码集合的并集(并集中的任意两个 Dewey 码都不会存在祖先-子孙的关系,这就避免了 ILE 中由 *removeAncestor* 操作造成的处理消耗),最后的结果为{“0.0.1”,“0.0.0.0”}.

需要说明的是,上述叙述中没有给出将 Dewey 码集合排序的示意,也没有给出求解两个 Dewey 集合的交集

运算示意.但是,第 3 节中的实验结果显示,直接以 Dewey 码形式参与运算的 LISA I 算法的性能并不突出,其原因正是由于 Dewey 码集合的排序、交集运算都较为耗时的缘故.这也是在 LISA II 中进一步吸收利于排序的简单数据类型的出发点.

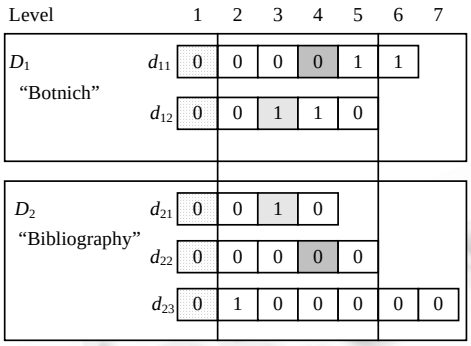

Fig.2 Dewey codes for LISA I

图 2 Dewey 码示例(LISA I)

2.1.2 LISA II 算法处理示例

有鉴于直接基于 Dewey 码数据类型的 LISA I 算法的不足,我们尝试将 Dewey 码转换为整数序列的处理方式.对应于图 2 中的形式,图 3 给出了整数序列的示意.序列中某层的整数是由原 Dewey 码对应该层的前缀映射而来的.例如,d_{11} 中第 2 层的整数“1”由图 2 中相应 Dewey 码的第 2 层前缀 Dewey 码“0.0”映射而来;而 d_{22} 中的“10”则来自于图 2 中相应 Dewey 码的第 5 层前缀 Dewey 码“0.0.0.0.0”.从 Dewey 码到整数的映射方法并不固定,只要映射能够保证不同 Dewey 码映射的整数必然不同,而相同 Dewey 码必然对应相同的整数即可.图 3 中的整数映射来自于针对树结构的前序遍历[4].

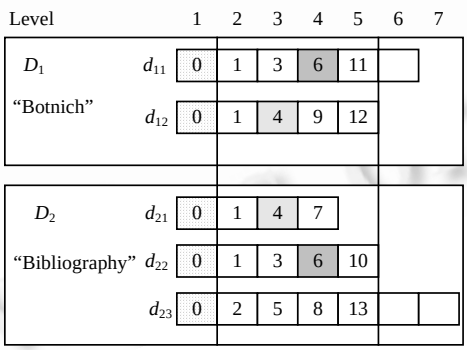

Fig.3 Integer sequence for LISA II

图 3 转换后的整数序列(LISA II)

在映射的过程中,可以利用“Dewey 码的长度必然属于$(2,\min\max L(D_1,D_2,\ldots,D_k)-1)$内”这一规律来缩减需要转换的前缀的数目.例如,根据示例数据可知,$\min\max L(D_1,D_2)-1=5$ 时,Dewey 码中长度超过 5 的前缀就不必进行转换.比如 d_{23} 的第 6 层、第 7 层前缀就不参与整数的转换,图 3 中 d_{23} 的第 6 层、第 7 层处的小框为空.

在完成将 Dewey 码转化为整数序列后,LISA II 求解 SLCA 节点的方式即与 LISA I 算法相同.首先从第 5 层开始,此时, $D_1'=\{11,12\}$, $D_2'=\{10,13\}$,两整数集合的交集为空,表明在第 5 层没有 SLCA 节点;之后进入第 4 层, $D_1'=\{6,9\}$, $D_2'=\{6,8\}$,交集为{6},那么在转换中,对应整数 6 的 Dewey 码就是第 4 层 SLCA 节点的 Dewey 码.后续几层的运算与此类似,不再赘述.

由于受篇幅所限,这里仅给出 LISA II 算法的描述,参见算法 1.其中的 *Inter*()函数表示两个有序整数集合的

交集运算,其描述由算法 2 给出.

算法 **1**. 求解 SLCA 问题的 LISA II32.

Input:

Dewey code set D_i for each keyword k_i

Output:

The SLCA list

1: **Transform** all Dewey codes into integer sequences, still **store** them according to keywords, marked as IS_i

2: **Compute** $\max L_i(IS_i)$ for each IS_i

3: **minmax*L***=$\min\{\max L_i | 1\le i\le k\}$

4: v={};tmp={}

5: **For** j=minmax$L$−1 **to** 2

6: **While** no IS_i is null **do** {

7: **Build** integer set D_i' $(1\le i\le k)$ corresponding level j and **sort** it, D_1' is the smallest set

8: $tmp = D_1'$

9: **For** i=2 **to** k

10: tmp=*Inter*(v, D_i');

11: v+=tmp;

12: **Delete** integer sequences, which have integers contained in v, from each IS_i

13:}

14:**Return** Dewey codes corresponding to v

算法 **2**. 有序整数集合的交集运算.

Input:

Two ordered integer set S_1,S_2

Output:

Intersect set of S_1,S_2

1: C_1=$|S_1|$;C_2=$|S_2|$

2. v={}

3: L_1=0;L_2=0

4: **While** (L_1<C_1 **&&** L_2<C_2){

5: **If** ($S_1[L_1]$=$S_2[L_2]$)

6: **Insert** $S_1[L_1]$ into v;

7: L_1+=1;L_2+=1;

8: **Else If** ($S_1[L_1]$<$S_2[L_2]$)

9: L_1+=1;

10: **Else**

11: L_2+=1;

12:}

13:**Return** v.

2.2 LISA II 算法复杂度分析

首先给出有序整数集合交集运算的复杂度.有如下命题:

命题 **1**. 给定两个有序整数集合 D_1' 和 D_2' .元素的最大值分别为 $\max_1$ 和 $\max_2$.如下选取两者中“较小”的集合 D_s':

$$D_i' = \begin{cases} D_1', & |D_1'| = |D_2'| \text{ and } \max_1 \le \max_2 \text{ or } |D_1'| < |D_2'| \\ D_2', & \text{Otherwise} \end{cases}.$$

定义整数集合 $med_{12}=\{p | p\in D_2' \wedge p\le \max_1\}$,那么,基于算法 2 的两个有序整数集合的交集运算复杂度为

$O(\max\{|D'_s|,|med_{12}|\})$.

证明:省略.

由 LISA II 的算法描述可知,在将对应各关键字的 Dewey 编码转换为整数序列的集合后(时间代价为常数),LISA II 算法的处理依赖于逐次计算各层节点集合的交集,而每一层的计算可分为 4 个步骤:构建对应各关键字的整数集合 D'_i;之后,对 D'_i 进行排序,在此,可以直接利用已有的成熟的排序算法;第 3 步是求解 D'_i 的交集;最后一步是将包含有交集元素的各路径节点序列从路径节点集合中删除.

对构建和排序阶段而言,最坏的情况是根本没有 SLCA 节点.这是因为,由于没有 SLCA 节点,后续的 IS_i 集合中元素的数目不会减少,在每一层循环中,算法 1 就必须扫描全部的 IS_i 集合得到$\{D'_i|1\le i\le k\}$,并将所有的 D'_i 集合进行排序.那么,单层循环中构建和排序的代价为

$$O\left(\sum_{i=1}^{k}(|D'_i|+|D'_i|\log|D'_i|)\right).$$

式中,$|D'_i|\log|D'_i|$来自集合排序的研究结论.

对于交集运算阶段,没有 SLCA 节点并不是它的最坏情况,仅仅比较最小的两个集合 D'_1 和 D'_2 就可以判断出这一层循环有没有共同元素的结论.根据命题 1,交集不为空时的运算代价为

$$O\left(\sum_{i=2}^{k}(\max\{|D'_1|,|med_{1,i}|\})\right).$$

式中, D'_1 为集合势最小的整数集合.

由于删除阶段的代价相较于其他阶段要小,故在如下的复杂度公式中忽略掉.综合上述讨论,算法 1 的运行复杂度的上界为

$$O\left((\min\max L-2)\left\{\sum_{i=1}^{k}(|D'_i|+|D'_i|\log|D'_i|)+\sum_{i=2}^{k}(\max\{|D'_1|,|med_{1i}|\})\right\}\right).$$

3 实 验

为了检验本文提出的 LISA 算法的性能,除了 LISA I,LISA II 算法外,本文还实现了文献[2]中的 ILE 算法.考虑到实验的可比性,本文实现的 ILE 算法与文献[2]中的有所不同,本文所提出的 ILE 算法采用的 B+树是基于内存构建的.

3.1 实验环境

3.1.1 实验数据

不论是 XML 数据随机生成工具(如 XMark 或 IBM 的 XMLGenerator),还是研究者常采用的标准 XML 数据集(如 Sigmod Record,Shakespeare 或 DBLP 等),都难以方便地控制数据集中 SLCA 节点的数目,所以,本文的实验数据采取了人工合成实验数据的方法:首先选取一个感兴趣的样本,即一个满足给定关键字组的 XML 数据片段(例如,可以采用图 1 的 XML 数据片段);之后,通过修改相应的 Dewey 编码得到实验的数据集.

下面结合图 1 的 XML 片段对具体的生成方法作一个简单介绍.

1. 根据图 1 所示的片段,得到 5 个感兴趣的节点的 Dewey 编码,分别为
   title→0.0.0.0.0;author#1→0.0.0.0.1.0;author#2→0.0.0.0.1.1;year→0.0.1.0 和 country→0.0.1.1.0;
2. 如果需要的数据量为 n,那么,首先复制源 Dewey 编码并修改 Dewey 编码的第 2 项,得到基本的片段,之后通过如下两种随机方式拓展新的基本片段:
   a) 随机交换 Dewey 码与关键字的对应关系;
   b) 在基本片段中随机的位置插入一个节点.

例 1:title→0.1.0.0.0;author#1→0.1.0.0.1.0;author#2→0.1.0.0.1.1;year→“**0.1.1.1.0**”和 country→“**0.1.1.0**”,即为修改第 2 项为“1”,并交换“year”与“country”的 Dewey 码后得到的测试片段.

例 2:title→0.2.0.0.0;author#1→“**0.2.0.0.1.0.0**”;author#2→“**0.2.0.0.1.0.1**”;year→0.2.1.0 和 country→0.2.1.1.0,即为在图 1 中“article”与“authors”间插入一个新节点生成的测试片段.

根据以上描述,既可以保证生成的测试数据易于按照要求生成足够的数据,又能够保证 XML 片段结构的多样性.实验中,采取了各关键字对应数据集的记录数目相同的设定.当然,也可以类似文献[2]中有关实验的设置.

需要指出的是,在固定数据集规模的前提下,当选取关键字的数目不同时,最终结果中的数据量与原始数据量的比值(也就是文献[2]中的频率参数)是变化的.例如,当设定数据总量为 6 000 条记录时,两个关键字的频率为 3000/6000=50%,而 5 个关键字时的比值为 1200/6000=20%.

3.1.2 调试环境

所有的实验都是在 Dell Dimension 8100 PC 机上实现的,机器的 CPU 为 Pentium IV,主频为 1.4GHz,物理内存为 256MB.所用的操作系统为 Windows 2000 专业版.所有的算法都使用 Java 语言实现,所用的 JDK 为 Java2 SDK 1.4 标准版.

实验采取了两种策略:一是固定关键字数目,变动数据集大小的方式;二是固定总体数据量,变动关键字的数目.

3.2 实验结果

3.2.1 内存空间与数据准备时间

从内存空间占用量来说,3 种算法中的 BPT/ILE 最低,即使在固定关键字为 5,变动实验数据从 2000/keyword 增加到 20000/keyword 的实验中,内存空间占用量变动仍不大,从 12MB 到 40MB;而 LISA I,LISA II 算法的内存占用相近,在固定关键字为 5,变动实验数据从 2000/keyword 增加到 20000/keyword 的实验中,内存空间占用量从 13MB 增大到 130MB.

至于 3 种算法的数据准备时间(InitData)——BPT/ILE 的关键字 Dewey 集合的获取以及内存 B+树的构建;LISA I 的关键字 Dewey 集合的获取、分解与排序以及 LISA II 算法的关键字 Dewey 集合的获取、转换与排序,在图 4(固定关键字数为 5,并递增关键字节点集合的数目)中给出.

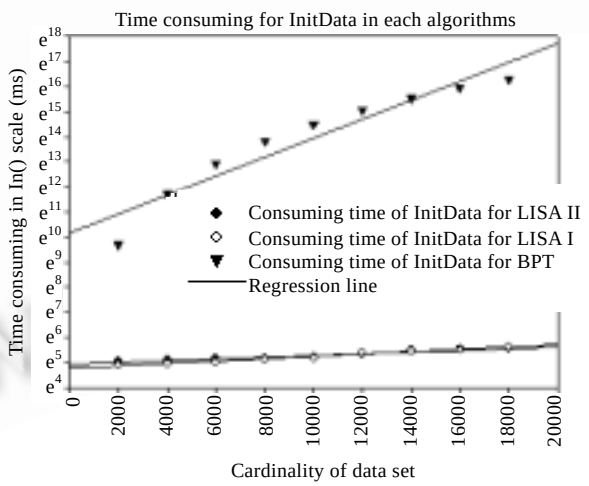

Fig.4 Time consuming for InitData in algorithms

图 4 数据预处理的时间代价图

由图 4 中可以看出,LISA II 和 LISA I 的数据准备时间远远小于 BPT/ILE 算法;而且,当固定关键字数目为 2 时,BPT/ILE 算法的数据准备时间往往难以接受,故在后续的固定关键字数目为 2 的 BPT/ILE 实验(如图 5、图 6 所示),以及保持数据总量、变更关键字数目的 SLCA 实验(如图 8 所示)中,与其他图相比均缺少数据点.

3.2.2 SLCA:固定关键字数目,变动数据集规模

图 5(固定关键字(2),变更节点数目)、图 6(固定关键字(5),变更节点数目)分别是固定关键字数目为 2 和 5 条件下的实验.从中可以看出,LISA II 算法具有最佳的性能,而且对数据量的变化比较平缓,而 BPT/ILE 和 LISA I 算法则具有明显的正比关系.

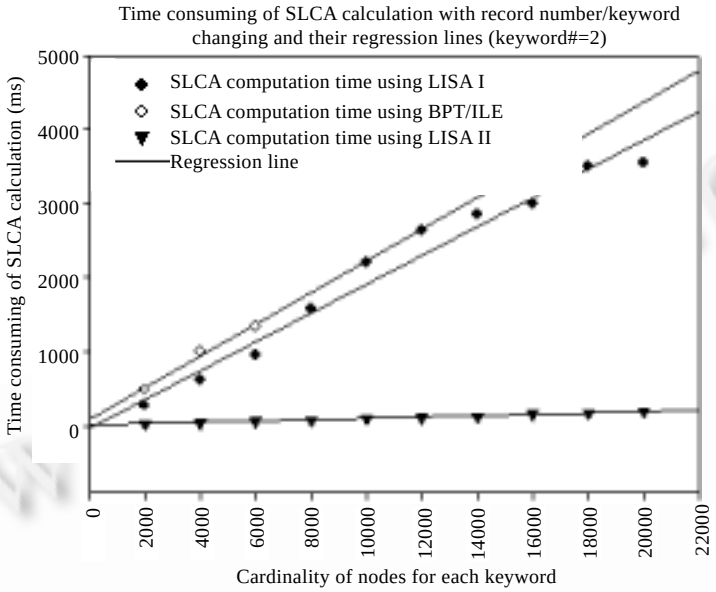

Fig.5 Time consuming for SLCA (keyword#=2)

图 5 SLCA 计算代价图(keyword#=2)

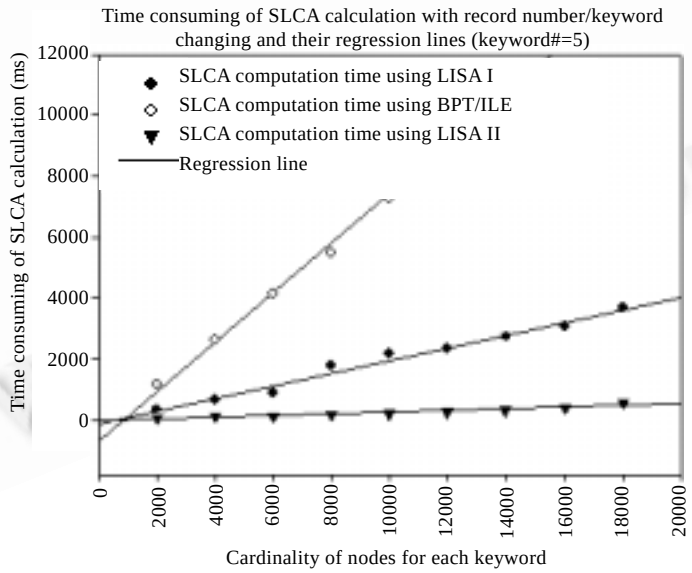

Fig.6 Time consuming for SLCA (keyword#=5)

图 6 SLCA 计算代价图(keyword#=5)

3.2.3 SLCA:固定数据集规模,考察不同关键字数目下的算法性能

图 7~图 9(同时变更关键字和总节点的数目)分别对应 3 种算法的实验结果.从图中可以看出,LISA II 算法具有最佳的表现.在数据总量保持 6 000 时,LISA II 算法下最大的时间值小于 50ms;而 BPT/ILE 的最大时间值为

800ms;LISA I 为 500ms.而且,从图中还可以看出,在数据量固定为某一值的条件下,随着关键字数目的增大,LISA II 算法的运算时间有下降的趋势,而其他两种算法则有相反的变化.

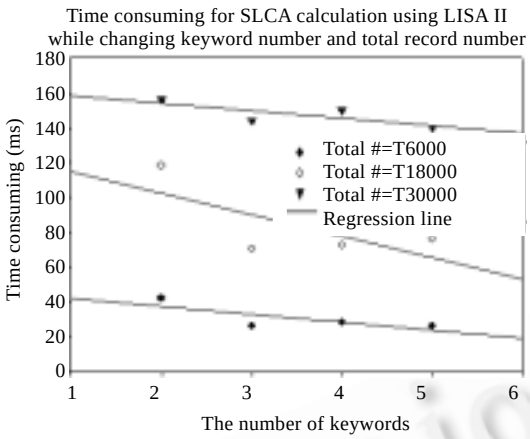

Fig.7 Time consuming for LISA II

图 7 基于 LISA II 算法的 SLCA 计算代价图

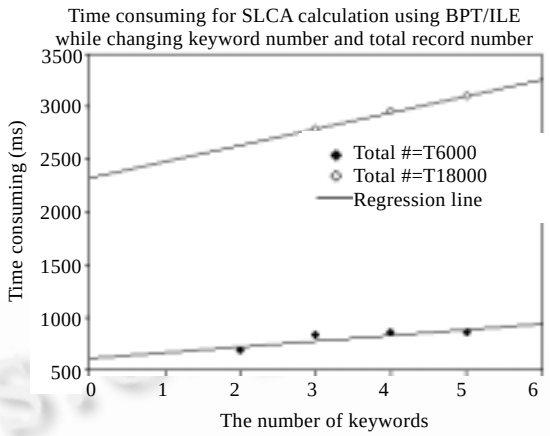

Fig.8 Time consuming for SLCA using BPT/ILE

图 8 基于 BPT/ILE 算法的 SLCA 计算代价图

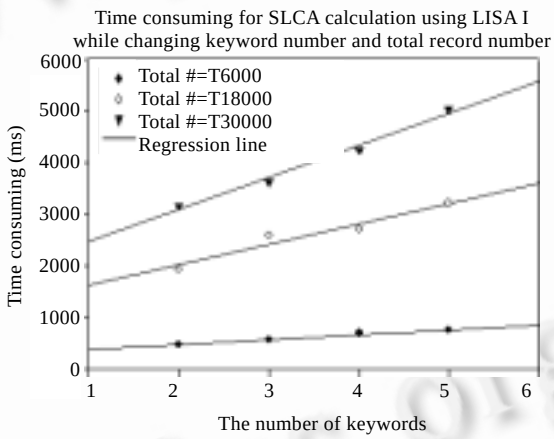

Fig.9 Time consuming for LISA I

图 9 基于 LISA I 算法的 SLCA 计算代价图

4 结论与研究展望

有别于基于 XML 查询语言的 XML 数据处理,XML 信息检索的处理方式以其便于普通用户使用的特点,而在最近几年受到广泛关注.其中的 SLCA 问题,即根据输入的关键字集合得到最紧密的 XML 数据片段,由于是 XML 信息检索处理中的基本问题,因而得到了深入的研究.文献[2]中给出了 3 种计算的算法:ILE,SE 和 Stack 算法,并声明 ILE 具有较好的表现.ILE 算法采用支持(keyword,dewey 码)类型的 B+树保存整个 XML 数据的节点信息;之后,通过反复执行 B+树上的 *lm*,*rm* 操作以及 Dewey 码的匹配运算求解 SLCA 节点,性能并不理想.与之不同,本文给出了仅仅依赖于对应关键字的 Dewey 码集合层次求解 SLCA 节点的 LISA 思路,即从深至浅逐层求解所在层上的 SLCA 节点.本文给出了两种基于这一思路的算法,分别为 LISA I 和 LISA II.二者的不同在于:前者保留 Dewey 码形式参与全部的 SLCA 计算;而后者首先将 Dewey 码转换为整数序列,从而依赖整数操作计算 SLCA 节点.实验证明,LISA II 算法具有良好的性能.另外,虽然本文的算法描述完全基于 Dewey 编码形式,但是分层求解 SLCA 节点的思路同样可以利用其他编码形式,例如文献[17]中的 PbiTree 编码.

后续研究将着眼于既能让用户表达出对目标 XML 片段所应有的结构信息,又能避免用户必须了解实际的 XML 数据组织结构的查询表达方式及其实现机制.随着 Web 时代的到来,信息检索作为方便普通用户使用的工具,在数据查询中具有突出的便利性.当 XML 数据成为当前数据存储、表示的主要形式时,在针对 XML 数据的查询处理中吸收信息检索的特点,成为未来 XML 数据管理手段中的一个研究点.但是,现有 XML 信息检索研究都具有不同的缺陷:扩展 XML 查询语言的方式从根本上没有摆脱不适于普通用户使用的弱点[8−10];要求用户提供 XML 数据片段的样例来指导后续的 XML 数据查询,因而不具有通用性[18];而仅仅将信息检索中的关键字查询扩展到 XML 数据,则由于不能够捕捉 XML 数据结构方面的信息,从而提高了结果集合计算的复杂度.为此,研究既能让用户表达出对目标 XML 片段所应有的结构信息,又能避免用户必须了解实际的 XML 数据组织结构的查询表达方式,就成为我们后续研究的方向.

另外一个值得深入研究的方向也与 XML 数据上的关键字查询紧密相关,即对获取的 XML 片段与查询模式紧密程度的判断.传统信息检索中关键字查询处理研究的一大特点就是能够将获得的文档根据与关键字查询模式的紧密程度进行排序,并将最为贴近关键字查询模式的文档展现给用户.当将这一概念延伸至 XML 数据时,由于 XML 数据的半结构化特点,使得这种相似程度判断不可避免地需要考虑 XML 片段上的结构特征.但是,现有的处理方式[19−22]在这一点上的表现仍然差强人意:要么过多地依赖于标签或节点的相对位置,要么不能有效地区分 XML 片段间的结构差异.因此,这一方面的研究还有待完善.

致谢　感谢稿件评审专家提出的中肯的意见和建议,使我们受益良多.

References:

[1] Meng XF, Zhou LX, Wang S. State of the art and trends in database research. Journal of Software, 2004,15(12):1822−1836 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/15/1822.htm

[2] Xu Y, Papakonstantinou Y. Efficient keyword search for smallest LCAs in XML databases. In: Ozcan F, ed. Proc. of the ACM SIGMOD Int'l Conf. on Management of Data (SIGMOD). Baltimore: ACM Press, 2005. 537−538.

[3] Tatarinov I, Viglas SD, Beyer K, Shanmugasundaram J, Shekita E, Zhang C. Storing and querying ordered xml using a relational database system. In: Franklin MJ, Moon B, Ailamaki A, eds. Proc. of the 2002 ACM SIGMOD Int'l Conf. on Management of Data (SIGMOD). Madison: ACM Press, 2002. 204−215.

[4] Kong LB, Tang SW, Yang DQ, Wang TJ, Gao J. XML indices. Journal of Software, 2005,16(12):2063−2079 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/16/2063.htm

[5] Clark J, DeRose S. XML path language (XPath) version 1.0 w3c recommendation. World Wide Web Consortium, 1999. http://www.w3.org/TR/xpath

[6] Barg M, Wong RK. Structural proximity searching for large collections semi-structured data. In: Proc. of the ACM Conf. on Information and Knowledge Management (CIKM 2001). Atlanta: ACM Press, 2001. 175−182.

[7] Cohen S, Mamou J, Kanza Y, Sagiv Y. Xsearch: A semantic search engine for XML. In: Freytag JC, Lockemann PC, Abiteboul S, Carey MJ, Selinger PG, Heuer A, eds. Proc. of the 29th Int'l Conf. on Very Large Data Bases (VLDB). Berlin: Morgan Kaufmann Publishers, 2003. 45−56.

[8] Curtmola E, Amer-Yahia S, Brown P, Fernàndez M. GalaTex: A conformant implementation of the XQuery FullText language. In: Florescu D, Pirahesh H, eds. Informal Proc. of the 2nd Int'l Workshop on XQuery Implementation, Experience, and Perspectives (XIME-P). Baltimore: ACM Press, 2005. http://www.galaxquery.com/galatex/

[9] Amer-Yahia S, Botev C, Shanmugasundaram J. TeXQuery: A FullText search extension to XQuery. In: Feldman SI, Uretsky M, Najork M, Wills CE, eds. Proc. of the 13th Conf. on World Wide Web (WWW). New York: ACM Press, 2004. 583−594.

[10] Amer-Yahia S, Lakshmanan LV, Pandit S. FleXPath: Flexible structure and full-text querying for XML. In: Weikum G, König AC, Deßloch S, eds. Proc. of the ACM SIGMOD Int'l Conf. on Management of Data (SIGMOD). Paris: ACM Press, 2004. 83−94.

[11] Fuhr N, Großjohann K. XIRQL: A query language for information retrieval in XML documents. In: Croft WB, Harper DJ, Kraft DH, Zobel J, eds. Proc. of the 24th Annual Int'l ACM SIGIR Conf. on Research and Development in Information Retrieval (SIGIR). New Orleans: ACM Press, 2001. 172−180.

[12] Florescu D, Kossmann D, Manolescu I. Integrating keyword search into XML query processing. In: Albert V, *et al*., eds. Proc. of the 9th Int'l World Wide Web Conf. Amsterdam: ACM Press, 2000. 119−136.

[13] Balmin A, Papakonstantinou Y, Hristidis V. A system for keyword proximity search on XML databases. In: Freytag JC, Lockemann PC, Abiteboul S, Carey MJ, Selinger PG, Heuer A, eds. Proc. of the 29th Int'l Conf. on Very Large Data Bases (VLDB). Berlin: Morgan Kaufmann Publishers, 2003. 1069−1072.

[14] Weigel F, Meuss H, Schulz KU, Bry F. Content and structure in indexing and ranking XML. In: Amer-Yahia S, Gravano L, eds. Proc. of the 7th Int'l Workshop on the Web and Databases (WebDB). Paris: ACM Press, 2004. 67−72.

[15] Guo L, Shao F, Botev C, Shanmugasundaram J. XRANK: Ranked keyword search over XML documents. In: Halevy AY, Ives ZG, Doan A, eds. Proc. of the 2003 ACM SIGMOD Int'l Conf. on Management of Data (SIGMOD). San Diego: ACM Press, 2003. 16−27.

[16] Botev C, Shanmugasundaram J. Context-Sensitive keyword search and ranking for XML. In: Doan AH, Neven F, McCann R, Bex GJ, eds. Proc. of the 8th Int'l Workshop on the Web & Databases (WebDB 2005). 2005. 115−120.

[17] Wang W, Jiang HF, Lu HJ, Yu JX. PBiTree coding and efficient processing of containment joins. In: Dayal U, Ramamritham K, Vijayaraman TM, eds. Proc. of the 19th Int'l Conf. on Data Engineering (ICDE). Bangalore: IEEE Computer Society, 2003. 391−402.

[18] Carmel D, Maarek YS, Mandelbrod M, Mass Y, Soffer A. Searching XML documents via XML fragments. In: Proc. of the 26th Annual Int'l ACM SIGIR Conf. on Research and Development in Information Retrieval (SIGIR). Toronto: ACM Press, 2003. 151−158.

[19] Amer-Yahia, S, Koudas N, Marian A, Srivastava D, Toman D. Structure and content scoring for XML. In: Böhm K, Jensen CS, Haas LM, Kersten ML, Larson P, Ooi BC, eds. Proc. of the 31st Int'l Conf. on Very Large Data Bases (VLDB). Trondheim: ACM Press, 2005. 361−372.

[20] Kailing K, Kriegel HP, Schönauer S, Seidl T. Efficient similarity search for hierarchical data in large databases. In: Bertino E, Christodoulakis S, Plexousakis D, *et al*., eds. Advances in Database Technology—EDBT 2004, the 9th Int'l Conf. on Extending Database Technology (EDBT). Heraklion: Springer-Verlag, 2004. 676−693.

[21] Yang R, Kalnis P, Tung AK. Similarity evaluation on tree-structured data. In: Ozcan F, ed. Proc. of the ACM SIGMOD Int'l Conf. on Management of Data (SIGMOD). Baltimore: ACM Press, 2005. 754−765.

[22] Augsten N, Böhlen MH, Gamper J. Approximate matching of hierarchical data using *pq*—Grams. In: Böhm K, Jensen CS, Haas LM, Kersten ML, Larson P, Ooi BC, eds. Proc. of the 31st Int'l Conf. on Very Large Data Bases (VLDB). Trondheim: ACM Press, 2005. 301−312.

附中文参考文献:

[1] 孟小峰,周龙骥,王珊.数据库技术发展趋.软件学报,2004,15(12):1822−1836. http://www.jos.org.cn/1000-9825/15/1822.htm

[4] 孔令波,唐世渭,杨冬青等.XML 数据索引技术.软件学报,2005,16(12):2063−2079. http://www.jos.org.cn/1000-9825/16/2063.htm

孔令波(1974－),男,山东日照人,博士生,主要研究领域为关系数据库实现技术,XML 数据处理技术,数据挖掘.

王腾蛟(1973－),男,博士,副教授,CCF 高级会员,主要研究领域为数据库,数据仓库,Web 数据集成,数据挖掘.

唐世渭(1939－),男,教授,博士生导师,CCF 高级会员,主要研究领域为数据库,半结构化数据,Web 数据集成,数据挖掘.

高军(1975－),男,博士,副教授,主要研究领域为数据库,数据仓库,半结构化数据,Web 数据集成,移动数据挖掘.

杨冬青(1945－),女,教授,博士生导师,CCF 高级会员,主要研究领域为数据库,数据仓库,Web 数据集成,移动数据挖掘.