第 35 卷第 7 期 通 信 学 报 Vol.35 No. 7
2014 年 7 月 Journal on Communications July 2014
doi:10.3969/j.issn.1000-436x.2014.07.006

面向 XML 关键字查询的高效 RKN 求解策略

陈子阳[1,2]，王璿[1,2]，汤显[3]
（1. 燕山大学 信息科学与工程学院，河北 秦皇岛 066004；
2. 河北省计算机虚拟技术与系统集成重点实验室，河北 秦皇岛 066004；3. 燕山大学 经济与管理学院，河北 秦皇岛 066004）

摘 要：构建结果子树是 XML 关键字查询处理的核心问题，其中求解与每个子树根节点相关的关键字节点是影响结果子树构建效率的重要步骤。针对已有方法不能正确求解基于 ELCA(exclusive lowest common ancestor)语义的相关关键字节点(RKN，relevant keyword node)的问题，提出 RKN 的形式化定义及相应的 RKN-Base 算法。该算法通过顺序扫描每个关键字节点一次即可正确判断其是否为某个 ELCA 节点的 RKN。针对 RKN-Base 不能避免处理无用节点的问题，提出一种优化算法 RKN-Optimized，该算法基于每个 ELCA 节点求其 RKN 集合，从而避免了对无用节点的处理，降低了时间复杂度。最后，通过实验验证了所提算法的高效性。

关键词：可扩展标记语言；子树构建；ELCA；相关关键字节点
中图分类号：TP311 **文献标识码**：A **文章编号**：1000-436X(2014)07-0046-10

Efficiently computing RKN for keyword queries on XML data

CHEN Zi-yang[1,2], WANG Xuan[1,2], TANG Xian[3]
(1. School of Information Science and Engineering, Yanshan University, Qinhuangdao 066004, China;
2. Key Laboratory for Computer Virtual Technology and System Integration of Hebei Province, Yanshan University, Qinhuangdao 066004, China;
3. School of Economics and Management, Yanshan University, Qinhuangdao 066004，China)

Abstract: Subtree results construction is a core problem in keyword query processing over XML data，for which computing the set of relevant keyword nodes (RKN) for each subtree's root node will greatly affect the overall system performance. Considering that existing methods cannot correctly identify RKN for ELCA semantics，the definitions of RKN and the RKN-Base algorithm were proposed，which can correctly judge whether a given node is an RKN of some ELCA node by sequentially scanning the set of inverted lists once. As RKN-Base cannot avoid processing all useless nodes，an optimized algorithm，namely RKN-Optimized，was then proposed，which computes RKN sets based on the set of ELCA nodes, rather than the set of inverted lists as RKN-Base does. As a result，RKN-Optimized avoids processing useless nodes, and reduces the time complexity. The experimental results verified the efficiency of the proposed algorithms.

Key words: XML; subtree results construction; ELCA; RKN

1 引言

随着 XML（extensible markup language，可扩展标记语言）应用领域的不断扩展，以 XML 格式表示和存储的数据量不断增大。作为一种简单易用的信息检索机制，基于 XML 数据的关键字检索技术得到了研究者的广泛关注[1~16]，其中的核心问题之一是如何高效返回满足特定语义的查询结果[3,5~11]。

通常情况下，把一个 XML 文档看成一棵带有节点标注信息的文档树 T。给定一个关键字查询 Q，每个查询结果 t_v 为 T 中满足查询条件的结果子树，t_v 包含 Q 中的所有关键字且 t_v 的根节点 v 满足特定

收稿日期：2014-04-07；**修回日期**：2014-06-20
基金项目：国家自然科学基金资助项目（61040023,61272124, 61303040）；河北省教育厅研究计划基金资助项目（Y2012014）；河北省科学技术研究与发展计划科技支撑计划基金资助项目（11213578）
Foundation Items: The National Natural Science Foundation of China (61040023, 61272124, 61303040); The Research Funds From Education Department of Hebei Province (Y2012014); The Science and Technology Research and Development Program of Hebei Province (11213578)

语义如 SLCA[6~10]，ELCA[3, 10~11]等。现有的子树构建方法[12~14]涉及 3 个步骤。步骤 1：遍历倒排表中所有节点，求解相应的 SLCA/ELCA 节点集。步骤 2：再次遍历倒排表，求得与每个 SLCA/ELCA 节点 v 相关的关键字节点集 R_v。步骤 3：对于每一个 R_v，构建满足特定约束条件的结果子树[12~14]。目前，XML 关键字查询处理的研究主要集中在如何提高步骤 1 的处理效率[3,5~11]，实际上，步骤 2 和步骤 3 是影响系统整体性能的关键因素。

以 MaxMatch 算法[12]为例，在 582 MB 的 XMark 注1数据集上运行 96 个查询，这些查询依据其结果选择率注2分成 6 组。图 1 给出基于不同算法[7~9]求解 SLCA 时，步骤 1 耗费时间占总时间的百分比。图 2 给出步骤 2 和步骤 3 所耗时间的比例关系。根据图 1 和图 2，有如下观察结果。

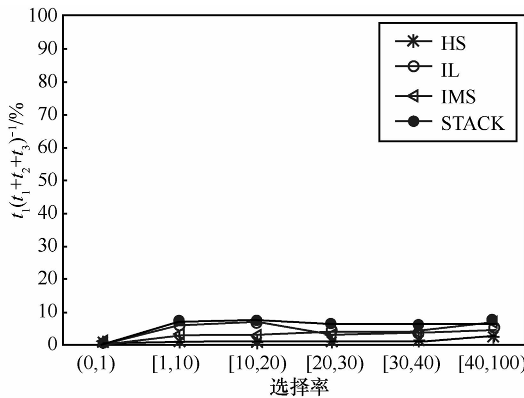


图 1　步骤 1 耗费时间(t_1)占总时间(t_1+t_2+t_3)的百分比

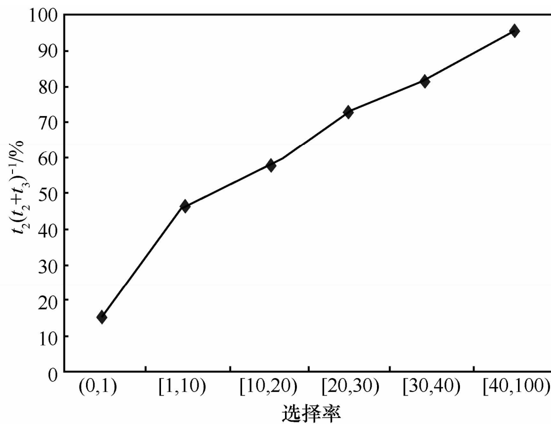


图 2　步骤 2 和步骤 3 所耗时间的比例关系

观察 1：相对于总时间(t_1+t_2+t_3)而言，步骤 1 所耗时间(t_1)几乎可以忽略不计(小于 10%)。

观察 2：当结果选择率较低时，步骤 3 耗费时间(t_3)较多，随着结果选择率的增加，步骤 2 耗费时间(t_2)所占比重逐渐增加。

通过上述的 2 个观察不难发现，无论步骤 1 采用何种算法，系统整体性能始终受限于步骤 2 和步骤 3。本文重点研究基于 ELCA 语义如何提高步骤 2 的处理效率。原因有 2 个方面：1)已有研究[17]表明，50%的关键字查询是探索式查询，即用户不知其具体查询目的。和 SLCA 相比，求解 ELCA 能够返回更多有意义的结果[6,11,15]，有助于用户发现所需信息。2)已有方法[13]的步骤 2 不能正确求解与每个 ELCA 节点相关的关键字节点集，导致结果子树可能包含无用信息或丢失有用信息。当然，本文方法适用于 SLCA 语义。

图 3 表示 XML 文档 D 对应的文档树 T。假设要从 T 中查找 Yanshan 大学的 Tom 在 Computer 期刊上发表的有关 XML 的文章，可以使用关键字查询 Q = {Yanshan，Tom，Computer，XML} 来完成该查询任务。根据 ELCA 语义，满足条件的子树根节点为 2 和 7。在判断节点 2 是否为 ELCA 节点时，节点 6、18、20、21 是被排除在外的，即对节点 2 来说，节点 6、18、20、21 是无关节点，因此 2 的相关关键字节点集为 R_2={3，4，22，23}。然而文献[13]得到节点 2 的相关节点集为 R_2={3，4，22，23，6，18，20，21}。如果用户想要得到匹配子树[14]，则用 R_2={3，4，22，23，6，18，20，21}构建以节点 2 为根的子树时，将包含无用信息(节点 5，6，16，18，19，20，21)，且丢失有用信息（节点 4 和 23）。

本文的主要贡献如下。

1) 提出相关关键字节点（RKN，relevant keyword node）的形式化定义。

2) 提出一种时间复杂度为 $O(d\sum_{i=1}^{m}|L_i|\log|ELCASet|)$ 的 RKN-Base 算法，其中 d 为给定的 XML 文档树深度，$ELCASet$ 为给定查询 Q = {k_1，k_2，…，k_m}的 ELCA 节点集，L_i 为 k_i 的倒排表。该算法通过顺序扫描每个节点一次即可正确判断其是否为某个 ELCA 节点的 RKN。

3) 针对 RKN-Base 算法不能避免处理无用节点的问题，提出优化算法 RKN-Optimized。该算法基于每个 ELCA 节点求其相关关键字节点，可以避免访问无用节点，将时间复杂度降为 $O(dm|ELCASet|\cdot$

注 1　http://monetdb.cwi.nl/xml。

注 2　选择率定义为结果个数和最短倒排表的比值。

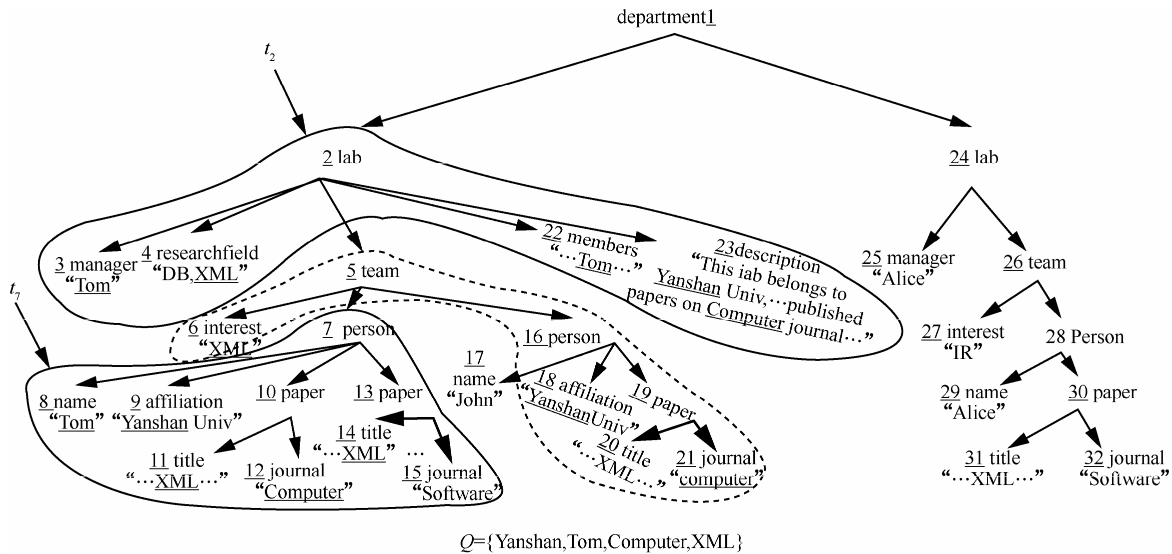


图 3　XML 文档 D 对应的文档树 T(每个节点旁边的数字是其 ID 编码，该编码是按照文档顺序排序的)

$\log|L_m|)$，其中，L_m 为最长倒排表。

4) 通过实验对算法的性能进行了验证和分析。

2　背景知识及相关工作

2.1　数据模型

本文用带有节点标注信息的树表示一个 XML 文档，其中节点表示元素或者属性，边表示节点间的直接嵌套关系。给定查询 Q，若节点 v 的标签、属性或者 v 的文本包含 Q 中的关键字 k，则称 v 直接包含 k，v 为关键字节点。

为了加速查询处理，通常需要给 XML 文档树中的每个节点 v 指定一个可以唯一标识的编码，用于确定节点间的位置关系，已有方法通常使用 Dewey[16]编码来标识节点。给定节点 v，Dewey 编码由其父亲节点的 Dewey 编码和其本身在兄弟节点中的顺序值组成。如图 3 所示，给每个节点一个 ID 值，该值等于以先序遍历方式访问 D 时该节点的访问次序。相应地，图 3 中每个节点 v 的 Dewey 编码由根节点到 v 的路径上所有节点的 ID 构成，称这种由节点 ID 构成的 Dewey 编码为 IDDewey 编码。

后续讨论中，除非有歧义，本文不对一个节点的 ID 值及其编码进行区分。例如，图 4 中的节点 9，即为节点 1，2，5，7，9。对于给定的 2 个节点 u 和 v，其位置关系包括：文档顺序($\prec_d$)，相等关系(=)，祖先后代关系($\prec_a$). $u \prec_d v$ 表示在文档中，u 位于 v 的前面，即节点 u 的 ID 值小于节点 v 的 ID 值，称 u 小于 v，反之称 u 大于 v。$u \prec_a v$ 表示 u 是 v 的祖先节点。如果 u 和 v 表示同一个节点，则 $u = v$，并且 $u \preceq_d v$ 和 $u \preceq_a v$ 同时成立。

2.2　查询语义及符号

给定查询 $Q = \{k_1, k_2, \cdots, k_m\}$，XML 文档 D 及 k_i 的倒排表 L_i，L_i 中的编码按照文档顺序升序有序。图 3 中查询 Q = {Yanshan，Tom，Computer，XML}的倒排表如图 4 所示，图 4 中的 Pos 表示倒排表中每个编码的下标位置。假设 $lca(v_1, v_2, \cdots, v_m)$表示节点 $v_1, v_2, \cdots, v_m$ 的最低公共祖先(LCA, lowest common ancestor)，则 Q 在 D 上的 LCA 集合定义为 $LCASet= LCA(Q) = \{v \mid v = lca(v_1, v_2, \cdots, v_m), v_i \in L_i\ (1 \leqslant i \leqslant m)\}$。例如图 3 中满足 Q ={Yanshan，Tom，Computer，XML}的 LCA 节点为 1，2，5，7。

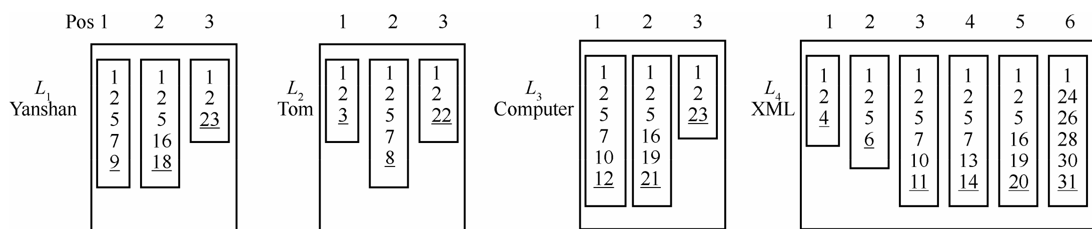


图 4　查询关键字 "Yanshan" (L_1)，"Tom" (L_2)，"Computer" (L_3)，"XML" (L_4)的倒排表

给定 LCA 节点 v，若去掉以 v 的后代 LCA 节点为根的子树后，以 v 为根的子树仍然包含所有的关键字，则称 v 为 ELCA 节点。其形式化定义为 $ELCASet= ELCA(Q)=\{v \mid \exists v_1 \in L_1,\cdots,v_m \in L_m (v=lca(v_1,v_2,\cdots,v_m) \wedge \forall i \in [1,m],\ \nexists x(x \in LCA(Q) \wedge child(v,v_i)) \preceq_a x)\}$，其中 $child(v,v_i)$ 指从节点 v 到节点 v_i 路径上 v 的孩子节点。例如图 3 中满足 Q = {Yanshan, Tom, Computer, XML}的 ELCA 节点为 2 和 7。

给定节点 u 和 v，若 u 和 v 具有祖先后代关系，则函数 $desc(u,v)$ 返回二者中的后代节点；若任一节点为空，则返回非空节点；若二者均空，则返回空值。假设 S 是有序节点集（按文档顺序升序有序），则 $lm(u，S)(rm(u，S))$ 返回 S 中比 u 小（大）的最大（小）节点；若不存在满足条件的节点，则返回空值。

2.3　相关工作

现有方法构建的结果子树主要分为 3 类：1）受限结果子树[12]；2）完全子树[2,8]；3）路径子树[15]。其中完全子树指以满足特定语义节点 v 为根且未经修剪的原始子树；路径子树是指以 v 为根且由 v 到全部关键字节点的路径组成的子树；受限子树是指以 v 为根且满足相关特殊删减条件的子树。目前研究较多集中于受限结果子树，代表性的算法有 MaxMatch[12]、MergeMatching[14]以及 RTF[13]。

MaxMatch 与 MergeMatching 算法只针对 SLCA 语义求解相关关键字节点并构建结果子树，不能适用于 ELCA 语义。RTF 算法虽然基于 ELCA 语义，但是不能正确求解相关关键字节点，导致其结果子树可能包含无用信息或者丢失有用信息。本文所提算法既能适用于 SLCA 语义也能适用于 ELCA 语义，并且能够正确、高效求解相关关键字节点。

3　RKN 及其性质

针对已有方法[13]不能正确求解基于 ELCA 语义的相关关键字节点集问题，为了进一步规范相关关键字节点，本文提出了相关关键字节点的形式化定义。

定义 1　相关关键字节点：对于给定的查询 Q，假设 v 为 Q 的 ELCA 节点，若关键字节点 u 满足以下条件：1) u 不是 LCA 节点且 u 是 v 的后代节点；2) v 到 u 的路径上不存在其他 LCA 节点，则称 u 为 v 的相关关键字节点。v 的所有相关关键字节点组成的集合称为 v 的相关关键字节点集(RKNS, relevant keyword node set)R_v。

例如，给定图 3 中查询 Q，$ELCASet=\{2, 7\}$。依据定义 1，节点 4 为节点 2 的包含关键字“XML”的后代节点，且从 2 到 4 的路径上不存在 LCA 节点，故 4 是 2 的 RKN。虽然节点 6 也为节点 2 的包含关键字“XML”的后代节点，但是从 2 到 6 的路径上存在 LCA 节点 5，根据定义 1，节点 6 不是节点 2 的 RKN。以此类推，节点 2 的 RKNS 为 $R_2=\{3, 4, 22, 23\}$，节点 7 的 RKNS 为 $R_7=\{8, 9, 11, 12, 14\}$。

依据 RKN 的定义可知，若要判断一个节点 u 是否为节点 v 的 RKN，则必须保证从 v 到 u 的路径上不存在 LCA 节点。针对这个重要的条件，提出了最近 LCA(CLCA，closest LCA)的概念。

定义 2　最近 LCA：给定关键字节点 u，$u_l=lm(u，ELCASet)$($u_r=rm(u，ELCASet)$)表示节点 u 在 $ELCASet$ 中的左(右)匹配节点，$u_l^a(u_r^a)$ 为 u 与 $u_l(u_r)$ 的 LCA 节点，则 u 的 CLCA 节点 c 定义为 $c=desc(u_l^a,u_r^a)$。

直观上看，节点 u 的 CLCA 节点 c 是给定查询 Q 的 LCA 节点，且从 c 到 u 的路径上不存在其他 LCA 节点，即 c 为距离 u 最近的祖先 LCA 节点。以图 3 中查询 Q 为例，$ELCASet=\{2，7\}$，关键字节点 6 在 $ELCASet$ 中的左匹配为节点 2，右匹配为节点 7，其 u_l^a 为 2，u_r^a 为 5，故其 CLCA 节点为 5。同理，关键字节点 18 在 $ELCASet$ 中的左匹配为节点 7，右匹配为空，其 u_l^a 为 5，u_r^a 为空，故其 CLCA 节点为 5。基于定义 2，用定理 1 来判断给定关键字节点是否为某个 ELCA 节点的 RKN。

定理 1　设 c 为给定关键字节点 u 的 CLCA 节点，若 c 为 ELCA 节点，则 u 是 c 的 RKN。

证明　因为 c 为 u 的 CLCA，所以 $c \preceq_a u$，且从 c 到 u 的路径上不存在其他 LCA 节点。若 c 为 ELCA 节点，根据定义 1，u 为 c 的 RKN。

例如，关键字节点 6 的 CLCA 为 5，但 5 不是 ELCA 节点，故 6 不是任一 ELCA 节点的 RKN。又如，节点 11 的 CLCA 节点为 7，且 7 是 ELCA 节点，故 11 是 7 的 RKN。

4　RKN-Base 算法

4.1　RKNS 的表示

对于给定的查询 $Q=\{k_1, k_2, \cdots, k_m\}$，通常用集合 $R_v=\{u_1, u_2,\cdots, u_n\}$表示并存储节点 v 的全部 RKN 节点编码。显然，RKN 节点编码已经存在于倒排表中，采用 $R_v=\{u_1, u_2,\cdots, u_n\}$表示并存储会产生节点

编码重复存储问题，造成内存空间的极大浪费。由于 v 的 RKNS 中许多节点都是连续分布在倒排表上，因此将 v 的 RKN 节点通过其在倒排表 L_i 中的下标位置表示：$R_vL_i=\{[s_1, e_1], \cdots [s_j, e_j]\}(j\leqslant n)$，其中，$s$ 表示一组连续 RKN 节点在倒排表上的起始位置，e 表示一组连续 RKN 节点在倒排表上的结束位置。

例如 $R_7L_4 = \{[3, 4]\}$，ELCA 节点 7 在 L_4(如图 4 所示)上第一个 RKN 的位置为 3，最后一个 RKN 的位置为 4。即通过[3, 4]可知节点 7 的 RKN 的编码是 1, 2, 5, 7, 10, 11 和 1, 2, 5, 7, 13, 14。

4.2 算法描述

假设每个倒排表 L_i 都有一个关联的指针 C_i，C_i 指向 L_i 中的某个节点。后续描述中，在不引起歧义的情况下，C_i 可用于指代其所指节点。函数 advance(C_i)的作用是让 C_i 指向下一个节点。

算法 1 getRKNS-B(*ELCASet*)

1) sort all ELCA nodes of *ELCASet* in document order

2) **while**(¬*eof IL*()) **do**

3) i←argmin{C_m}

4) c←*desc*(*lca*(C_i，*lm*(C_i，*ELCASet*))，*lca*(C_i，*rm*(C_i，*ELCASet*)))

5) **if**(c∈*ELCASet*) **then** *update*(R_c. L_i)

6) *advance*(C_i)

7) **endwhile**

函数 eofIL()

1) **if**(∄i，such that C_i does not reach the end of L_i) **then return** TRUE

2) **else return** FALSE

算法 1 中，第 1 行对 *ELCASet* 中所有节点按照文档顺序排序。假设查询 Q 有 m 个关键字，在每次的循环过程中（第 2）~7）行），算法 1 先从 C_1 到 C_m 中选择最小节点 C_i（第 3）行），然后在第 4）行得到 C_i 的 CLCA 节点 c，如果 c 是 ELCA 节点，那么根据定理 1，C_i 是 c 的 RKN，更新 R_c. L_i（第 5）行）。第 6 行让 C_i 指向下一个节点。

例 1 给定图 3 中查询 Q = {Yanshan, Tom, Computer, XML}，*ELCASet* = {2, 7}，根据算法 1，由图 4 倒排表可知其关键字节点处理顺序为：3, 4, 6, 8, 9, 11, 12, 14, 18, 20, 21, 22, 23, 31。首先处理节点 3，其 CLCA 节点为 2，因为 2 为 ELCA 节点，则节点 3 是 2 的 RKN，于是得到 $R_2L_2=\{[1, 1]\}$(1, 2, 3 节点属于 L_2)。其次，处理节点 4，其 CLCA 节点为 2，则节点 4 是 2 的 RKN，于是有 $R_2L_4=\{[1, 1]\}$。再次，处理节点 6，其 CLCA 节点为 5，由于 5 不是 ELCA 节点，故节点 6 不是任一 ELCA 节点的 RKN，直接略过。后续的处理过程与前述相似，在处理完 14 个关键字节点后，ELCA 节点 2 的 RKNS 为：$R_2L_1 = \{[3, 3]\}$，$R_2L_2 = \{[1, 1], [3, 3]\}$，$R_2L_3 = \{[3, 3]\}$，$R_2L_4 = \{[1, 1]\}$; ELCA 节点 7 的 RKNS 为: $R_7L_1=\{[1, 1]\}$，$R_7L_2=\{[2, 2]\}$，$R_7L_3=\{[1, 1]\}$，$R_7L_4=\{[3, 4]\}$。

4.3 算法分析

给定查询 Q，算法 1 需要处理倒排表中的所有节点，对于每个节点第 3）行的处理代价是 $O(md)$，其中 d 为给定的 XML 文档树深度。第 4）行中 *lm* 和 *rm* 的代价是 $O(d\log|ELCASet|+3d)$，实际中，由于 3<<log|*ELCASet*|，所以第 4）行的代价是 $O(d\log|ELCASet|)$。第 5）行的代价是 $O(d\log|ELCASet|)$。因此算法 1 的时间复杂度为 $O(d\log|ELCASet|\sum_{i=1}^{m}|L_i|)$。

可见，虽然算法 1 可以正确求解 RKN 节点，但是当 RKN 的个数远远小于倒排表中的节点总数时，即 $\sum_{i=1,v_i\in ELCASet}^{|ELCASet|}|RKN(v_i)|<<\sum_{i=1}^{m}|L_i|$，算法 1 需要处理大量无用节点(非 RKN 节点)。

5 RKN-Optimized 算法

5.1 主要思想

不同于算法 1 遍历处理倒排表上的每一个节点，本文设计了一种优化算法，通过遍历处理每一个 ELCA 节点，利用 ELCA 节点去相应的倒排表查找其 RKNS 来提高算法效率。算法的主要思想为：对于每一个 ELCA 节点 v，首先计算其后代节点在每个倒排表上的区间范围，用 $vL_i = [s,e]$表示。其次寻找 v 的每个后代 LCA 节点 u，并计算 u 的后代节点在每个倒排表上的区间范围，以 $u.L_i = [s,e]$表示。最后，依据 RKN 定义，vL_i 减去 uL_i 即为节点 v 在第 i 个倒排表上 RKNS 的区间范围。

5.2 算法描述

算法 2 采用栈 S 存储每个 ELCA 节点的 IDDewey 编码中的数字，这更加容易发现 ELCA 后代节点中的 LCA 节点。如图 5(a)所示，当栈中存储节点 1, 2, 5, 7 的编码时，节点 5 即为 ELCA 节点 2 的后代 LCA 节点。算法 2 首先将所有 ELCA 节点按

文档顺序排序（第 1)行），然后依次处理每一个 ELCA 节点(第 2)~26)行)。具体来说，对于待入栈的 ELCA 节点 v，将栈 S 中不是 v 祖先的节点全部出栈（第 3)~6)行），对于每一个出栈的节点 u，若 u 是 ELCA 节点，则直接得到其 RKNS（第 5)行）。然后，将 v 尚未入栈的祖先节点依次入栈（第 7)~25)行）。入栈过程中，对于 v 的每个祖先节点 $u(u \neq v)$，则首先将 u 标记为非 ELCA 节点（第 9)行），然后检测当前栈顶元素 $top(S)$是否为 ELCA 节点，如果 $top(S)$ 是 ELCA 节点（第 10)行将返回 TRUE），则在第 11)行调用 *locateRange* 得到 u 的后代节点在各个倒排表中的范围 uL_i，在第 13)行将 uL_i 从 $top(S)L_i$ 中去除。当 v 的所有祖先节点入栈后，在第 17)~22)行处理 v。如果 $top(S)$为 ELCA(第 17)行)，则将 u 标记为 ELCA 节点（算法 2 第 17)~22)行 $u = v$），然后调用 *locateRange* 求出节点 u 的后代节点在各个倒排表中的范围 uL_i（第 18)行），然后将 u 的 RKNS 从 $top(S)$ 的 RKNS 中移除（第 19)~21)行）。在 24)行将 u 入栈。最后，在处理完所有的 ELCA 节点后，将栈中剩余节点全部出栈，求出栈中 ELCA 节点相应的 RKNS（27)~30)行）。

算法 2　getRKNS-O(*ELCASet*)

1) sort all ELCA nodes of *ELCASet* in document order
2) **foreach** ($v \in ELCASet$) **do**
3)　　**while** ($top(S) \not\prec_a v$) **do**
4)　　　　$u \leftarrow pop(S)$
5)　　　　**if** ($u.isELCA$ = TRUE) **then** output R_u
6)　　**endwhile**
7)　　**foreach** (u on the path from $top(S)$ to v) **do**
8)　　　　**if** ($u \neq v$) **then**
9)　　　　　　$u.isELCA \leftarrow$FALSE
10)　　　　　　**if** ($top(S).isELCA$ = TRUE) **then**
11)　　　　　　　　*locateRange*(u);
12)　　　　　　　　**foreach** ($i \in [1, m]$) **do**
13)　　　　　　　　　　$R_{top(S)}.L_i \leftarrow R_{top(S)}.L_i - R_u.L_i$
14)　　　　　　　　**endfor**
15)　　　　　　**endif**
16)　　　　**else**
17)　　　　　　**if** ($top(S).isELCA$ = TRUE) **then**
18)　　　　　　　　$u.isELCA \leftarrow$TRUE; *locateRange*(u);
19)　　　　　　　　**foreach** ($i \in [1, m]$) **do**
20)　　　　　　　　　　$R_{top(S)}.L_i \leftarrow R_{top(S)}.L_i - R_u.L_i$
21)　　　　　　　　**endfor**
22)　　　　　　**endif**
23)　　　　**endif**
24)　　　　*Push*(S，u)
25)　**endfor**
26) **endfor**
27) **while**($\neg isEmpty(S)$) **do**
28)　　$u \leftarrow pop(S)$
29)　　**if** ($u.isELCA$ = TRUE) **then** output R_u
30) **endwhile**

函数 **locateRange(*u*)**

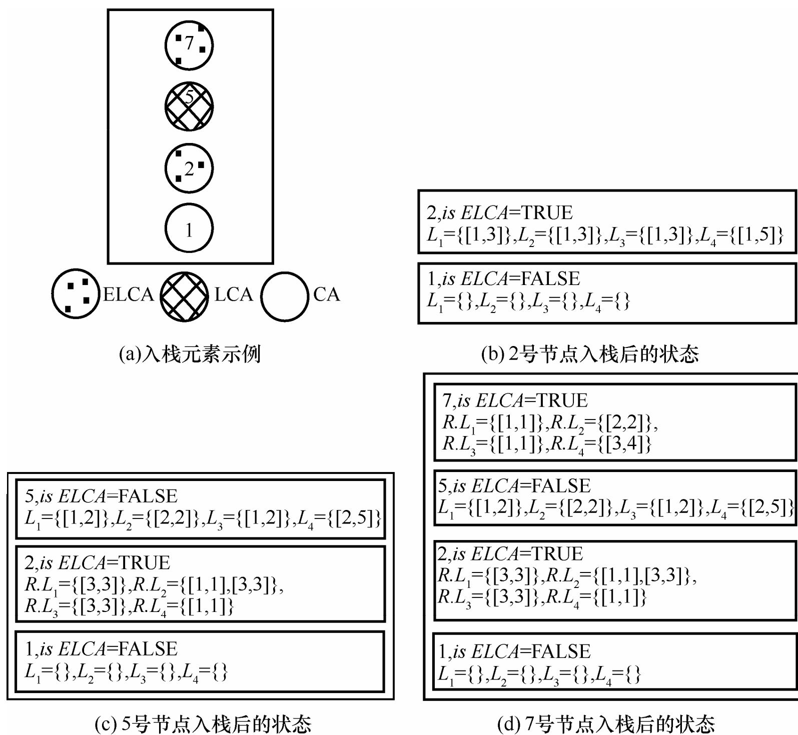
1) **foreach** ($i \in [1, m]$) **do**
2) $x \leftarrow$ the position of the *first* descendant node of u in L_i
3) $y \leftarrow$ the position of the *last* descendant node of u in L_i
4) $R_u.L_i \leftarrow \{[x, y]\}$
5) **endfor**

例 2　给定图 3 中查询 Q =｛Yanshan, Tom, Computer, XML｝及其 $ELCASet = \{2, 7\}$，算法 2 首先处理节点 2，将编码 1.2 直接入栈并初始化，如图 5(b)所示。其次处理节点 7，因为栈顶节点 2 为 ELCA 节点，5 为 LCA 节点，故需求出节点 5 的后代节点在倒排表中的范围并将节点 5 入栈。入栈的同时，将栈顶节点 2 的后代范围减去节点 5 的后代节点范围，如图 5(c)所示。然后将节点 7 入栈，求出其后代节点范围。由于此时 *ELCASet* 中再无其他节点，故 ELCA 节点处理完毕，如图 5(d)所示。最后将栈中节点全部出栈，最终得到 ELCA 节点 2 和 7 的 RKNS。

5.3　算法分析

直观上看，算法 2 依次处理 *ELCASet* 中每一个 ELCA 节点，从而得到其 RKNS。由于每一个 ELCA 节点最多调用 2 次 locateRange 函数，而 locateRange 函数的代价为 $O(dm\log|L_m|)$，因此算法 2 的时间复杂度为 $O(|ELCASet|dm\log|L_m|)$。显然，和算法 1 相比，算法 2 可以避免处理无用节点。

由于不同 SLCA 节点之间没有祖先后代关系，其 RKN 集合的求解非常简单，只需要顺序访问每个 SLCA 节点，在倒排表上调用 locateRange 函数，即可求出该 SLCA 节点的 RKN 区间范围，具体过程不再赘述。



(a)入栈元素示例　(b) 2号节点入栈后的状态

(c) 5号节点入栈后的状态　(d) 7号节点入栈后的状态

图 5　例 2 的运行图

6　实验

6.1　实验环境

本文实验所用机器的基本配置为 AMD AthonⅡX2 270 Professor 3.4 GHz CPU，2 GB 内存，500 GB 硬盘，操作系统为 Windows XP。

为了进行公平比较，只比较基于内存数据的算法性能。实验所采用的数据集为 XMark (582 MB)与 DBLP [注3](876 MB)。在解析 2 个数据集之后，基于 Oracle Berkeley DB4，使用一个散列文件来存储所有的关键字倒排表，其中散列文件的每个 key 是一个关键字 k，value 是 k 对应的存储 Dewey 编码的倒排表。当处理给定查询 Q 时，Q 对应的倒排表首先被读入到内存，所有实验结果均不考虑 I/O 代价的影响。

从 XMark 数据集中选取了一组关键字，这些关键字根据其在数据集中出现的次数分为 3 类：1) 低频关键字(100~1 000); 2) 中频关键字(10 000~40 000); 3) 高频关键字(300 000~600 000)。基于这些关键字，生成了 12 个查询，按照关键字个数分成 4 组(Group1：2 个关键字；Group2：3 个关键字；Group3：4 个关键字；Group4：5 个关键字)，如表 1 所示。$\sum_{i=1}^{m}|L_i|$ 表示所有关键字倒排表的长度之和，$|L_{\max}|$($|L_{\min}|$)表示最长(最短)倒排表的长度，N_E 表示满足条件的 ELCA 节点个数，$R_E=N_E/|L_{\min}|$表示结果选择率。DBLP 数据集上的查询也用类似的方式生成，由于 XMark 数据集包含较复杂的文档结构，且数据集大小可以按比例调整以便进行扩展性测试，本文中主要展示基于 XMark 数据集的实验结果。所有算法均用 Visual C++ 语言实现，运行时间为算法执行 100 次的平均值。

6.2　性能比较和分析

表 2 为 12 个查询的运行时间比较，其中，T_B 表示 RKN-Base 运行时间，T_O 表示 RKN-Optimized 运行时间，$R_e=T_O/T_B$ 表示 2 个算法运行时间的比率，由实验结果可知。

1) 对 RKN-Base 而言，由于其需要处理所有的关键字节点(表 1 第 3 列 $\sum_{i=1}^{m}|L_i|$)，因此当关键字节点个数较多时，性能较差，例如 Q10。同时，其性能也受 ELCA 节点个数(表 1 第 6 列 N_E)的影响，当 ELCA 节点个数变多时，所需时间明显增大，这是由其时间复杂度中的 log|*ELCASet*|决定的，例如 Q2 和 Q3。

2) 对 RKN-Optimized 而言，由于其循环次数由 ELCA 节点个数决定，因此当 ELCA 节点个数较少时，性能较好，如 Q1，Q4，Q7，Q8，Q9，Q10，

注 3　http://www.informatik.uni-trier.de/~ley/db/。

表 1　基于 XMark 数据集的查询

ID	Keywords	$\sum_{i=1}^{m}\|L_i\|$	$\|L_{max}\|$	$\|L_{min}\|$	N_E	R_E/%	Group
Q1	bidder, incategory	710 593	411 575	299 018	1	0.0003	Group1
Q2	bidder, text	827 825	528 807	299 018	54 200	18.13	
Q3	listitem, bold	675 087	370 118	304 969	117 933	38.67	
Q4	bidder, listitem, incategory	1 015 562	411 575	299 018	1	0.0003	Group2
Q5	bidder, date, emph	1 106 809	457 231	299 018	29 089	9.73	
Q6	check, listitem, keyword	693 390	352 121	36 300	11 552	31.82	
Q7	bidder, listitem, date, incategory	1 472 793	457 231	299 018	1	0.0003	Group3
Q8	check, listitem, keyword, date	1 150 621	457 231	36 300	8 087	22.28	
Q9	takano, keyword, bold, emph	1 089 928	370 118	17 129	6 448	37.64	
Q10	bidder, text, time, date, incategory	2 009 949	528 807	299 018	1	0.0003	Group4
Q11	order, keyword, text, emph, increase	1 552 894	528 807	16 700	1 918	11.49	
Q12	check, keyword, bold, text, date	1 744 577	528 807	36 300	16 204	44.64	

Q11 等，甚至在某些情况下优于 RKN-Base 4 个数量级，例如 Q1，Q4，Q7，Q10。随着结果选择率的增加，RKN-Optimized 的性能优势逐渐下降，个别情况下甚至比 RKN-Base 差，这是因为 RKN-Optimized 需要处理大量的 ELCA 节点，例如 Q3。

表 2　RKN-Base 与 RKN-Optimized 运行 12 个查询的运行时间

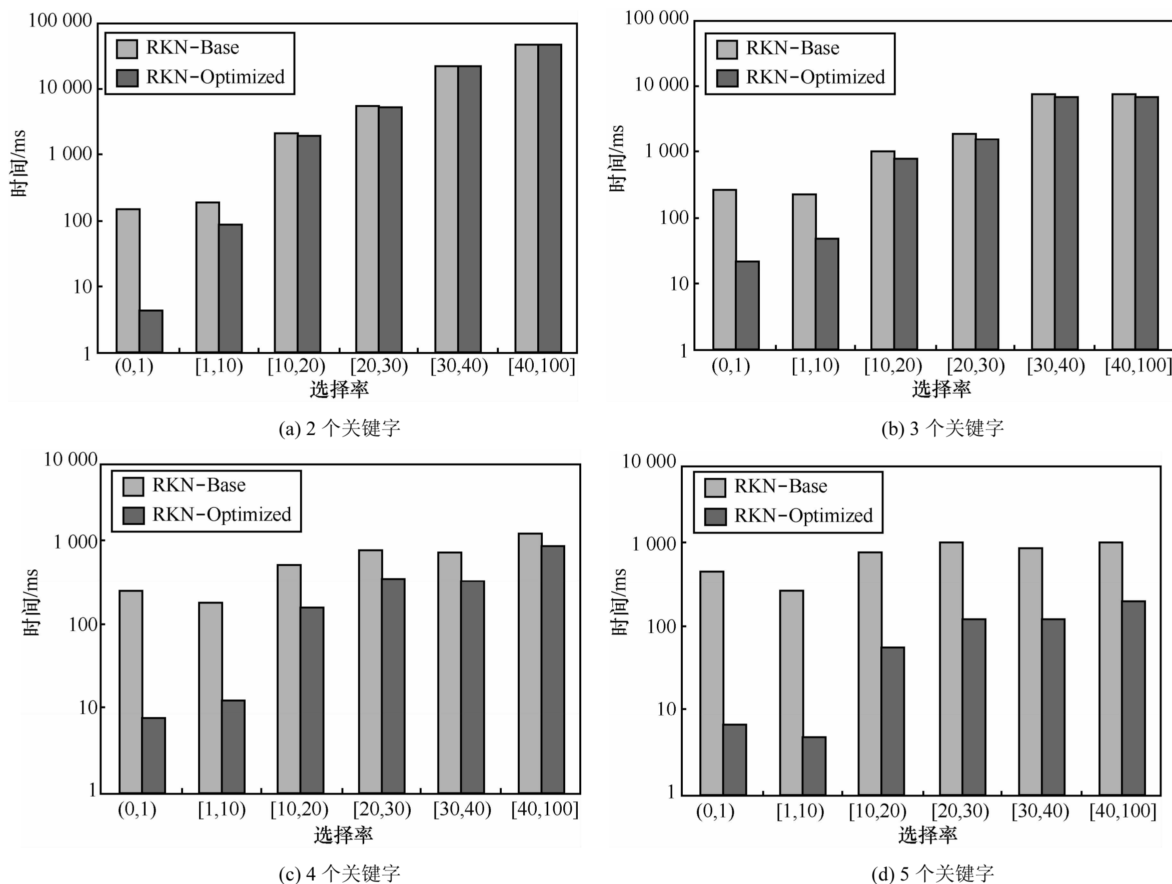
查询 *ID*	T_B/ms	T_O/ms	R_e/%
Q1	1 248	0.1	0.008
Q2	18 018	17 845	99.039
Q3	63 274	68 672	108.531
Q4	2 106	0.1	0.0047
Q5	3 978	3 588	90.196
Q6	1 279	1 124	87.881
Q7	4 227	0.1	0.002
Q8	1 841	499	27.104
Q9	1 592	437	27.449
Q10	7 895	0. 047	0.000 6
Q11	2 402	125	5.203
Q12	3 728	1 779	47.719

以上观察及表 2 的实验结果可进一步由表 3 的数据来解释。表 3 为 2 种算法中执行基本操作(编码比较操作)的次数，该数据可以客观反映不同算法运行时间的差异。其中，N_B 表示 RKN-Base 编码比较次数，N_O 表示 RKN- Optimized 编码比较次数，$R_e = N_O/N_B$ 表示 2 个算法编码比较次数的比率，如表 3 所示，对 Q1，Q4，Q7，Q10 而言，RKN-Optimized 的编码比较次数远少于 RKN-Base，因此其所需运行时间远小于 RKN-Base。对 Q3 而言，RKN-Optimized 的编码比较次数接近 RKN-Base，因此其所需运行时间与 RKN-Base 也相差不大。

表 3　RKN-Base 与 RKN-Optimized 运行 12 个查询的编码比较次数

查询 *ID*	$N_B\times10^3$	$N_O\times10^3$	N_e/%
Q1	2 543	0.04	0.0015 73
Q2	1 495 939	1 471 036	98.335 29
Q3	3 501 520	3 492 110	99.731 26
Q4	4 551	0.06	0.0013 18
Q5	247 065	213 349	86.353 39
Q6	54 119	34 976	64.627 95
Q7	7 910	0.08	0.001 011
Q8	51 528	17 030	33.049 99
Q9	38 391	7 768	20.233 91
Q10	12 751	0.101	0.000 792
Q11	42 979	1 225	2.850 229
Q12	124 417	67 445	54.208 83

除了表 1 的 12 个查询，本文还随机生成了 17 万查询，算法运行时间按照查询关键字个数及结果选择率分类，如图 6 所示。该实验结果进一步验证了：1) 关键字个数较少，结果选择率较高时，RKN-Optimized 性能最差。例如图 6(a)中结果选择率为[40,100]时，RKN-Optimized 与 RKN-Base 运行时间基本相同。2) 关键字个数较多，结果选择率较低时，RKN-Optimized 性能最好。如图 6(d)中结果选择率为(0,1)时，



(a) 2 个关键字 (b) 3 个关键字

(c) 4 个关键字 (d) 5 个关键字

图 6 运行时间比较

RKN-Optimized 运行时间优于 RKN-Base 近 100 倍。

3) 考虑单个因素时，随着关键字个数的增加，RKN-Optimized 性能优势逐渐上升。例如图 6 中随着关键字个数增加，图 6(c)和图 6(d)中 RKN-Optimized 的整体性能优于图 6(a)和图 6(b)。另外，随着结果选择率的升高，RKN-Optimized 的优势逐渐下降。

图 7 给出在不同大小的 XML 文档上执行查询 Q8 的运行时间。从图中可以看出本文方法有非常好的扩展性，对于其他的查询，有类似的结果，因此不再赘述。

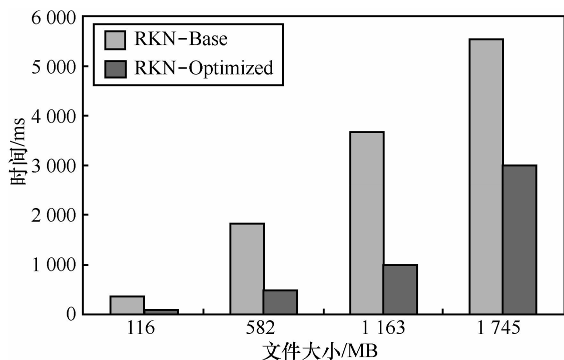


图 7 Q8 在不同大小 XML 文档上的运行时间

表 4 为 DBLP 数据集上的 10 个查询，其中，N_E 为 ELCA 个数，算法运行时间如图 8 所示。通过实验可知，RKN-Optimized 在大多数情况下优于 RKN-Base，原因同样是因为 RKN-Optimized 避免了处理大量无用节点。

表 4 基于 DBLP 数据集的查询

ID	Keywords	N_E
Q1	article, book	1 456
Q2	algorithm, article	18 349
Q3	data, article	26 611
Q4	article, database	5 753
Q5	XML, article	1 033
Q6	year, 2001	59 355
Q7	book, article, mining	6
Q8	algorithm, article, 2001	521
Q9	article, data, mining	1 563
Q10	data, XML, article	209

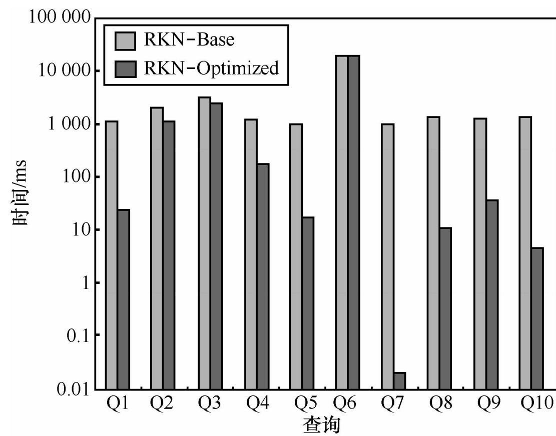


图 8　DBLP 数据集上运行时间

7　结束语

构建结果子树是 XML 关键字查询处理的核心问题，其中求解与每个子树根节点相关的关键字节点集是影响结果子树构建效率的重要步骤。针对已有方法不能正确求解基于 ELCA 语义的相关关键字节点集的问题，本文提出了相关关键字节点的形式化定义 RKN。依据该定义提出了基本算法 RKN-Base，该算法通过顺序扫描每个关键字节点一次即可判断其是否为某个 ELCA 节点的相关关键字节点。针对 RKN-Base 算法不能避免处理无用节点的问题，提出了优化算法 RKN- Optimized。该算法基于每个 ELCA 节点求其相关关键字节点，可避免处理无用关键字节点，降低时间复杂度。最后通过实验对所提算法的性能进行了验证和分析。

本文的后续工作将针对生成结果子树的第 3 步（构建结果子树）展开研究，进而设计并实现一个完整、高效的 XML 关键字查询处理系统。

参考文献：

[1] TATARINOV I, VIGLAS S, BEYER K S, *et al*. Storing and querying ordered XML using a relational database system[A]. SIGMOD Conference[C]. 2002. 204-215.

[2] GUO L, SHAO F, BOTEV C, *et al*. Xrank: ranked keyword search over XML documents[A]. SIGMOD Conference[C]. 2003.16-27.

[3] ZHOU RUI, LIU CHENGFEI, LI JIANXIN. Fast elca computation for keyword queries on XML data[A]. International Conference on Extending DB Technology[C]. Lausanne, Switzerland, 2010.549-560.

[4] COHEN S, MAMOU J, KANZA Y, *et al*. Xsearch: a semantic search engine for XML[A]. VLDB[C]. 2010. 45-56.

[5] LI G, FENG J, WANG J, *et al*. Effective keyword search for valuable lcas over XML documents[A]. CIKM[C]. 2007.31-40

[6] ZHOU J, BAO Z, CHEN Z, *et al*. Top-down SLCA computation based on list partition[A]. DASFAA[C]. 2012.

[7] WANG W Y, WANG X L, ZHOU A Y. Hash-search: an efficient slca-based keyword search algorithm on XML documents[A]. LNCS 5463[C]. 2009. 496-510.

[8] XU Y, PAPAKONSTANTINOU Y. Efficient keyword search for smallest lcas in XML databases[A]. SIGMOD Conference[C]. 2005.

[9] SUN C, CHAN C Y, GOENKA A K. Multiway slca-based keyword search in XML data[A]. WWW[C]. 2007.1043-1052.

[10] ZHOU J, BAO Z, WANG W, *et al*. Fast SLCA and ELCA computation for XML keyword queries based on set intersection[A]. ICDE[C]. 2012.

[11] XU Y, PAPAKONSTANTINOU Y. Efficient lca based keyword search in XML data[A]. EDBT[C]. 2008.

[12] LIU Z, CHEN Y Reasoning and identifying relevant matches for XML keyword search[J]. PVLDB, 2008. 1(1): 921-932.

[13] KONG L, GILLERON R, LEMAY A. Retrieving meaningful relaxed tightest fragments for XML keyword search[A]. EDBT[C]. 2009. 815-826.

[14] ZHOU J, BAO Z, CHEN Z, *et al*. Fast result enumeration for keyword queries on XML data[A]. DASFAA[C].2012.95-109.

[15] HRISTIDIS V, KOUDAS N, PAPAKONSTANTINOU Y, *et al*. Keyword proximity search in XML trees[J]. IEEE Trans Knowl Data Eng, 2006,18(4):525-539.

[16] TATARINOV I, VIGLAS S, *et al*. Storing and querying ordered XML using a relational database system[A]. Special Interest Group on Management of Data Conference[C]. Madison, USA, 2002. 204-215.

[17] BRODER A Z. A taxonomy of Web search[J]. SIGIR Forum, 2002, 36(2):3-10.

作者简介：

陈子阳（1973-），男，黑龙江五常人，博士，燕山大学教授、博士生导师，主要研究方向为数据库理论与系统等。

王璿[通信作者]（1977-），女，黑龙江齐齐哈尔人，博士，燕山大学副教授，主要研究方向为高性能计算、数据库等。E-mail: wangxuan@ysu.edu.cn。

汤显（1978-），女，山东潍坊人，博士，燕山大学讲师，主要研究方向为闪存数据库、信息检索等。