第36卷 第19期 计 算 机 工 程 2010年10月
Vol.36 No.19 Computer Engineering October 2010
·软件技术与数据库· 文章编号：1000—3428(2010)19—0075—03 文献标识码：A 中图分类号：TP311

一种高效的动态 XML 文档树编码机制

胡江明，李建华，杜章华，魏 锋
(中南大学信息科学与工程学院，长沙 410075)

摘 要：针对现有 XML 编码机制时空效率不高、对 XML 文档动态更新支持不够等问题，结合素数和 IBSL 2 种编码机制，提出一种新的 XML 文档树编码机制——基于素数的二进制字符串编码机制。理论分析和实验均证明该编码机制具有较好的查询效率，能够高效地支持 XML 文档更新操作，大幅降低空间成本。

关键词：XML 文档；更新；素数

Efficient Labeling Mechanism for Dynamic XML Document Trees

HU Jiang-ming, LI Jian-hua, DU Zhang-hua, WEI Feng
(Institute of Information Science and Engineering, Central South University, Changsha 410075, China)

【Abstract】There are two main problems about the existing labeling mechanism: low space-time efficiency and insufficient supporting for dynamic update of XML document. Combining with the prime and IBSL labeling mechanism, this paper presents a novel XML tree labeling called Prime-based Binary String Labeling(PBSL), which has high efficiency in query, can support update operation for XML document, and greatly reduce the storage space.

【Key words】XML document; update; prime

1 概述

随着 XML 相关标准的不断推广和应用，Web 上出现了大量的 XML 数据。XML 是一种专门为因特网设计的标记语言，目前已成为数据交换和存储领域事实上的标准。

大量 XML 文档的出现促进了 Native XML 数据库的研究，编码机制是其中热点之一。一般情况下，XML 树中的每个节点都被赋予唯一的编码，通过这个编码可以快速地确定任意 2 个节点之间是否存在父子、祖先后裔、兄弟等关系，这种为 XML 文档树中的节点分配唯一编码的具体策略称为编码机制。因此，XML 文档的节点编码机制是实现 Native XML 数据库高效查询的重要条件，对它的研究具有非常重要的意义。

为了优化 XML 查询求解，有效地适应 XML 数据的查询和更新，一个高效的编码机制应具有以下 3 个方面的特征：

(1)确定性。能够快速高效地确定 XQuery 中定义的 13 种 XML 节点间结构关系的轴值。

(2)动态性。能够支持 XML 数据的动态更新，而不用重新改变任何已经存在的编码。

(3)压缩性。节点编码所占的空间应该尽量小。

本文在对已有各种编码机制深入研究的基础上，提出了一种新的 XML 文档树编码机制——基于素数的二进制字符串编码(Prime-based Binary String Labeling, PBSL)机制。

2 相关研究

为了高效地查询求解，已经提出了多种 XML 文档编码机制。依据编码原理，分为基于区间的编码机制和基于路径的编码机制两大类。

2.1 基于区间的编码机制

有关文档编码的早期研究多集中在区间编码方法上，文献[1]是典型的基于区间的编码机制。基于区间的编码机制的主要思想是利用 XML 节点的有序特点，根据每一个元素节点的文档顺序位置为其赋予一个正整数三元组<start, end, level>或其变体形式<order, size, level>(后者可以理解为 start=order; end=order+size)。节点 A 是节点 B 的祖先的充要条件是 A.start≤B.begin 且 A.end≥B.end。节点 A 是节点 B 的父节点的充要条件是节点 A 是节点 B 的祖先且 A.level= B.level-1。

区间编码机制根据区间判断节点关系，其优点是能够有效地支持包含与文档位置有关的计算，节点编码简单，平均长度短；缺点是使用非等值进行节点关系判断，需要单独提供层次信息，且始终不能完全支持 XML 文档更新操作，尽管有许多人做了大量努力，提出了有预留机制的区间编码[2]。

2.2 基于路径的编码机制

基于路径的编码机制的主要思想是利用 XML 文档的嵌套特点，根据 XML 文档的嵌套结构，给从文档根节点及从它开始所能到达的每条路径的其他节点赋予一个编码。它主要包括素数相关编码机制[3]和基于前缀的编码机制[4-5]。

素数编码机制[3]的优点是采用整除运算判断祖先-后裔关系速度较快，对 XML 树的扇出不敏感，缺点是不能完全支持更新操作，编码长度较长。前缀整除编码(PDIV)机制是对素数编码机制的一种改进，其“个人标记”是可以重复利用的，优点是编码长度短，对 XML 树的扇出不敏感，缺点是不能完全支持更新操作。

IBSL[4]和 DeweyBFN[5]是至今少有的 2 种较好的能完全支持更新的动态 XML 树编码机制，它们的层标识都是变长

作者简介：胡江明(1981－)，男，硕士研究生，主研方向：XML 数据库，软件测试；李建华，教授、博士研究生；杜章华、魏 锋，硕士研究生

收稿日期：2010-03-07 **E-mail：**hjm1981@126.com

的二进制字符串，不同之处在于：IBSL 的层标识左右位置关系按二进制字符串的字典序比较，而 DeweyBFN 是按二进制字符串所表示的相对应的小数大小比较的，两者的缺点都是对 XML 文档的扇出敏感，编码长度较长。总之，本文的 PBSL 编码机制在完全避免重新编码的前提下，提高了祖先后裔的查询效率，同时有效缩减了空间成本。

3 PBSL 编码机制

3.1 基本定义

定义 1(素数列) 素数列 P_i，即 P_1=2, P_2=3,…。

定义 2(前缀整除关系与前缀约数) 2 个正整数 x、y 之间存在前缀整除关系当且仅当：

(1)$x|y$ 且 $x\neq y$；

(2)若 $z=y/x$，则 z 的任意素数因子 a 和 x 的任意素数因子 b 之间存在 $a\geq b$ 的关系。

此时，x 称为 y 的前缀约数。

定义 3(字典序) 给定 2 个连续的二进制字符串 S_1 和 S_2，其字典序如下：

$S_1 \equiv S_2 \Leftrightarrow S_1$ 和 S_2 完全相同；

若 S_1 当前位为“0”，而 S_2 相应的当前位为“1”，那么 $S_1 \prec S_2$；

若 S_1 的长度小于 S_2，S_1 是 S_2 的前缀，且 S_2 除去与 S_1 完全相同的前缀部分，剩下的位都是“1”，那么 $S_1 \prec S_2$；

若 S_1 的长度大于 S_2，S_2 是 S_1 的前缀，且 S_1 除去与 S_2 相同的前缀部分，剩下的位都是“0”，则 $S_1 \prec S_2$。

定义 4(PBSL 编码) PBSL 编码是由一个二进制字符串和一个素数(或素数乘积)组成的二元组<primes, bina_string>。

定义 5(最近公共祖先) 节点 C 是节点 A 和 B 的最近公共祖先当且仅当 C 被包含在 A 祖先和 B 祖先的交集中且其层次在交集中最高。

3.2 静态 PBSL 编码

任何 XML 文档都可以表示为节点带有标签的树型结构，称为 XML 树。而对 XML 文档进行编码，指的是按照一定的规则，为 XML 树中的每个节点赋予唯一的标识。

定义 6(静态 PBSL 编码) 静态 PBSL 编码是在 XML 文档载入阶段为 XML 树中的每个节点赋予一个唯一的 PBSL 编码，其规则如下：

根节点编码为 $<1,\text{null}>$；

XML 文档树的第 1 层的第 i 个子节点编码为 $<P_i, 1^i0>$ (1^i0 表示由 i 个“1”和“0”组成的二进制字符串)；

XML 文档树的其他节点编码：设节点 P 的最大素数因子为 P_k，则其第 i 个子节点的编码为 $<P.primes*P_{k+i-1}, 1^i0>$。

利用该规则对左边的 XML 文档片断进行编码，其结果如图 1 实线部分所示。

```
<Productset>
<Product1>
    <ID>1</ID>
    <Name>car</Name>
    <Price>100</Price>
</Product1>
<product3>
    <ID>3</ID>
    <Name>book</Name>
    <Price>10</Price>
</Product3>
</Productset>
```

```
<Product2>
<ID>2</ID>
<Name>shoes</Name>
<Price>20</Price>
</Product2>
```

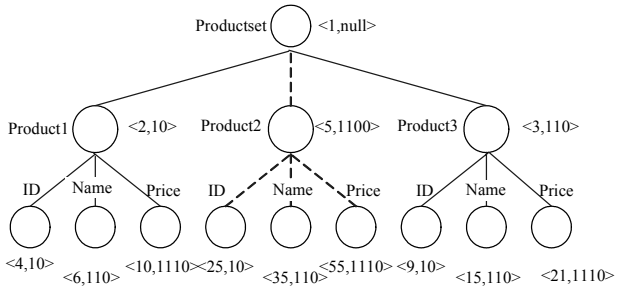


图 1 PBSL 编码示例

3.3 动态 PBSL 编码

XMI 动态性是指对 XML 文档的插入、删除和更新操作的支持。由于 XML 文档的删除和更新操作不涉及 XML 节点编码的改变，因此对于 XML 树编码机制来说，动态性就是能支持 XML 节点的插入。

完全支持 XML 节点的插入指的是在任何父节点的任何 2 个相邻孩子节点之间插入一个新的孩子节点时，可以得到一个合法的编码，而不用修改存在的任何节点编码。

算法 动态 BSPL 编码算法

输入 插入位置左节点编码 label(left)=<a,b>，插入位置右节点编码 label(right)=<c,d>

输出 插入新节点编码 label(new)=<e,f>

开始

如果插入新节点和左节点是兄弟，获取它们所有兄弟所用到的最大素数因子在素数列中的序号 id=idl;

如果插入新节点和右节点是兄弟，获取它们所有兄弟所用到的最大素数因子在素数列中的序号 id=idr;

如果和左右节点都不是兄弟，id=2;

(1)最左端插入

f=b ⊕ 0;

$e=a*P_{idl+1}$;

(2)中间插入

如果 b 的长度≤d 的长度，f=d ⊕ 0;

如果 b 的长度>d 的长度，f=b ⊕ 1;

$e=a*P_{idl+1}$;

(3)最右端插入

f=d ⊕ 1;

$e=a*P_{idr+1}$;

结束

算法的时间和空间复杂度均为 $O(1)$，利用该算法把右边的 XML 文档片断插入到左边片断元素节点 Product1 和 Product3 之间，其编码结果如图 1 虚线部分所示。

3.4 PBSL 编码机制性质分析

3.4.1 祖先后裔关系的快速查询

节点 A 与节点 B 之间存在祖先后裔关系当且仅当 A.primes 是 B.primes 的前缀约数。

证明：当节点 A、B 相差仅为一层时，上述命题显然成立。当节点 A、B 相差多层时，如果 A.primes 是 B.primes 的前缀约数，设 P 为 B.primes/A.primes 的最小素数因子，那么编码为 A.primes*P 的节点是 A 的子女节点；编码为 A.primes*P 的节点是 B 的祖先节点，因此，节点 A 与节点 B 之间存在祖先后裔关系。由此可见，判断节点之间的祖先后裔关系只需要判断 2 个节点的编码之间是否存在前缀整除关

系。从前缀整除关系的定义来看，上述判断的时间复杂度为 $O(n)$。

3.4.2　左右位置关系的判断

节点 A 和 B 左右位置关系的判断如下：首先找到 A 和 B 的最近公共祖先 C；再找到 C 孩子中 A 与 B 的祖先 A' 和 B'；接着判断 A' 和 B' 左右位置关系，即得到 A 和 B 的左右位置关系。

证明：节点 A' 和 B' 是在同一层上的兄弟节点，节点 A 是节点 A' 的后裔，节点 B 是节点 B' 的后裔，显然，节点 A 和 B 左右位置关系与节点 A' 和 B' 左右位置关系相同。

3.4.3　编码长度分析

假设最糟情况，XML 文档树是最大深度为 N 的满 K 叉树，编码长度单位为 bit。

PDIV 编码机制中，该文档树编码最长的节点是第 N 层最右端的节点，编码长度小于等于 $(1+1\mathrm{b}(\prod_{i=1}^{n-1} ik\ln(ik)))$。

IBSL 编码机制中，该 XML 文档树编码最大长度为 $(N-1)(K+1)$；本文的 PBSL 编码机制中，该 XML 文档树编码最大长度是 PDIV 与 IBSL 编码机制中 self_label 之和，即 $(1+1\mathrm{b}(\prod_{i=1}^{n-1} ik\ln(ik)))+(K+1)$

综上所述，得到图 2 和图 3，从图中可以清楚地看出 PBSL 充分利用了 PDIV 编码短的优点，与 IBSL 相比，其空间性能好很多，而且随着 XML 文档树的深度和扇出度的增加，优势更明显。

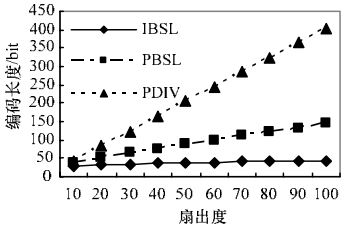


图 2　扇出度对编码长度的影响(深度=5)

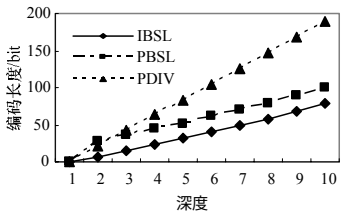


图 3　深度对编码长度的影响(扇出度=20)

3.5　扩展动态 PBSL 编码

当 XML 文档存在大量删除操作时，会出现一个问题：节点被删除后，其相应的编码也被删除，上述算法没有办法重新利用被删除的编码，这样编码长度会迅速增长，并且随着被删除节点的增多进一步恶化。为了避免这种情况的出现，可以对动态 PBSL 编码进行扩展，回收那些被删除的编码。

4　实验及性能分析

将本文提出的 PBSL 编码和 PDIV 及 IBSL 编码进行比较。所有实验都是在用 Java DOM4J 包和 Oracle 10g 的 PL/SQL 编写的程序下进行的。

表 1 给出的是实验用到的 3 个测试数据集的特性(限于篇幅，选取部分数据罗列)。其中，第 2 个数据集是文献[6]提供的；第 3 个是文献[7]提供的；第 1 个数据集则是由 XMark 自动生成的，其中，005 表示生成文件时参数 f 的大小为 0.005。表 2 给出了 3 种编码机制静态编码性能的比较结果，其中，Avg 和 Max 分别表示编码的平均和最大长度。为了测试 PBSL 编码机制的更新效率，本文选取表 1 中 reed.xml 作为插入文本片断，分别插入到 XMark_005.xml 第 2 层 3 个不同的位置，表 3 给出了 3 种编码机制的比较结果。

表 1　测试数据集

文档名称	深度 Max/Avg	扇出度 Max/Avg	节点数
XMark_005.xml	13/5.60	127/1.68	17 509
reed.xml	5/3.70	703/1.79	18 855
mvp_64k.xml	6/4.98	520/2.04	267 805

表 2　静态编码性能　bit

文档名称	PDIV		PBSL		IBSL	
	Avg	Max	Avg	Max	Avg	Max
XMark_005.xml	30.43	92.95	34.73	141.33	53.72	150
reed.xml	29.93	50.58	46.77	717.38	361.55	720
mvp_64k.xml	34.35	50.52	125.15	557.61	440.99	1033

表 3　动态编码性能　bit

插入位置	PDIV		PBSL		IBSL	
	Avg	Max	Avg	Max	Avg	Max
最左端	不支持	不支持	43.20	721.49	214.89	723.00
第 3 个节点后	不支持	不支持	43.20	721.49	216.44	726.00
最右端	32.39	92.95	43.20	721.49	217.48	728.00

综上所述，相对于 PDIV，PBSL 编码机制以稍多的空间代价换取了对更新操作的完全支持；而相对于 IBSL，无论在静态性能还是动态性能方面都更优越，而且随着 XML 文档节点的增加，优势越明显。

5　结束语

本文提出了一种基于素数的二进制字符串编码机制。理论分析和实验结果表明，该编码机制能够实现祖先后裔关系的快速查询，全面支持更新操作，编码长度也较短。在本文基础上，可以进一步展开其他研究工作，如本文提出的编码机制需要单独的素数例加以支持，如何减少由此带来的存储代价，以及如何在 PBSL 编码基础上设计和实现支持动态更新的 Native XML 数据库的索引结构。

参考文献

[1] Zhang Chun, Naughton J, Dewitt D, et al. On Supporting Containment Queries in Relational Database Managerment Systems[J]. ACM SIGMOD Record, 2001, 30(2): 425-436.

[2] Li Quanzhong, Moon B. Indexing and Querying XML Data for Regular Path Expression[C]//Proc. of the 27th VLDB Conference. Roma, Italy: [s. n.], 2001: 361-370.

[3] Wu Xiaodong, Lee Mong-Li, Hsu Wynne. A Prime Number Labeling Scheme for Dynamic Ordered XML Trees[C]//Proceedings of the 20th International Conference on Data Engineering. [S. l.]: IEEE Press, 2004.

[4] Ko Hye-Kyeong, Lee Sang-Keun. A Binary String Approach for Updates in Dynamic Ordered XML Data[J]. IEEE Transactions on Knowledge and Data Engeneering, 2008, 99(1): 602-607.

[5] Wang Chenying, Yuan Xiaojie, Wang Xin, et al. An Efficient Numbering Scheme for Dynamic XML Trees[C]//Proc. of 2008 International Conference on Computer Science and Software Engineering. Wuhan, China: [s. n.], 2008: 704-707.

[6] University of Washington. XML Repository[EB/OL]. (2008-12-10). http://www.cs.washington.edu/research/xmldatasets/.

[7] Niagara. NIAGARA Experimental Data[EB/OL]. (2008-09-08). http://www. cs.wisc.edu/niagara/data.html.

编辑　张正兴