一种 XML多分支路径索引查询算法

吉根林,肖 袁

(南京师范大学 数学与计算机科学学院,江苏 南京 210097)

[摘要] 为高效地实现 XML多分支路径查询,提出了基于索引的 XML多分支路径查询算法 depthjoin.首先对 XML文档进行编码并创建索引,然后对查询树进行查询匹配.采用栈存储多分支路径中的单路径,对多分支结点采用索引信息判定其子结点是否具有共同的祖先结点或父结点.与现有的 XML分支查询算法相比,算法 depthjoin充分利用索引,不需要进行单路径的连接操作,实验表明其查询效率比现有的查询算法高.

[关键词] XML查询, XML索引, XML多分支路径查询

[中图分类号] TP311 [文献标识码] B [文章编号] 1672-1292(2007)01-0063-05

A XML Multiple Branch Path Query Algorithm Based on Index

Ji Genlin, Xiao Yuan

(School of Mathematics and Computer Science, Nanjing Normal University, Nanjing 210097, China)

Abstract XML single path query methods and simple branch path query methods have been presented, but how to query multiple branch path is not solved well. This paper presents the algorithm depthjoin for querying XML multiple branch path based on index. The algorithm encodes the XML documents and creates an index for them. It restores the single path of the multiple branches path using stack on the process of matching of the query tree, judging if the branch nodes have the same ancestors or parent by the index. Compared with the existing algorithms, the algorithm does not need join the single paths. The query efficiency is superior to the existing algorithms.

Key words XML query, XML index, XML multiple branch path

0 引言

XML数据查询尤其是结构复杂的 XML数据查询是一项重要的研究课题. XML数据查询主要分为单路径查询和多分支路径查询.单路径查询语句形如 a//b/c;目前针对单路径的查询方法有 A(K)[1]、D(k)[2]、APEX[3]、1-index[4]、XISS[5]、DataGuide[6]等,其基本思路是对 XML文档树建立路径索引,有效地减少查询单路径的时间.解决多分支路径查询的方法可分为两种:一是首先将多分支路径查询拆分成由根结点到叶子结点的单路径,并通过结构连接算法[7,8]得到单路径的查询结果,然后将所得到的单路径查询结果进行连接操作得到分支路径查询结果.以多分支路径查询 a[b='c'] //[d='e']为例,首先将该分支路径进行拆分得到两个单路径 a//b/c和 a//d/e,然后利用结构连接算法分别得到这两个单路径的查询结果,最后将两个单路径的连接结果进行连接操作,得到最终满足条件的分支路径查询.二是将多分支路径拆分成结点对,分别对结点对进行查询,最终将结点对连接成多分支查询结果.以多分支路径查询 a[b='c'] //[d='e']为例,首先将其拆分成 a//b, b/c, a//d, d/e;然后分别对这些结点对进行查询,最后将查询的结果进行连接操作.方法一由于拆分操作产生了多条单路径,在将单路径连接成多分支路径的算法中采用多重循环比较,因此影响了查询效率.方法二将多分支路径拆分成结点对,将会得到大量的中间结果,在对这些中间结果进行连接操作时,同样因为多重循环比较影响执行效率.为了避免 XML多分支

收稿日期: 2006-11-30
基金项目: 江苏省高校自然科学基金(04KJB520075)资助项目.
作者简介: 吉根林(1964-),教授,博士生导师,主要从事数据库、数据挖掘与 XML技术等方面的教学与研究. E-mail: glj@njnu.edu.cn

查询过程中多条单路径的连接操作而带来的查询效率低的问题,本文提出了一种XML多分支索引查询算法depthjoin.该算法在对查询树进行查询匹配的过程中,对多分支路径中的单路径采用pathstack算法[9],而对分支结点,则利用索引信息判定其子结点是否具有共同的祖先结点或父结点,从而不需进行单路径的连接操作.实验表明,算法depthjoin的查询效率比现有的查询算法高.

1　XML编码与索引

算法depthjoin首先要对XML文档树中的结点进行编码,编码方法为:将XML文档看作是流文件,每个标签或内容按其出现的先后次序赋予一个位置编码,记标签L的起始编码为$begin(L)$,结束编码为$end(L)$,则标签L的编码为$[begin(L), end(L)]$.例如,图1中<PLAY>位置记为1,<TITLE>位置记为2,A1位置记为3,</TITLE>位置记为4,<ACT>位置记为5,直至文档最后</PLAY>位置为68,因此标签PLAY的编码为(1,68),标签TITLE的编码为(2,4)等等.对于图1所示的XML文档,其对应的文档树及其结点编码如图2所示.显然,祖先结点u的编码区间$[begin(u), end(u)]$包含后裔结点v的编码区间$[begin(v), end(v)]$.这种编码方案的优点是只需在解析文档的时候记录各标签的起始位置,即只需对XML文档进行一次遍历即可获得文档结点的编码.

推论　设XML文档树中的任意两个结点u和v是祖先-后裔关系,当且仅当$begin(u)<begin(v)<end(v)<end(u)$.特别地,若两个结点$u$和$v$是父子关系,当且仅当$begin(u)<begin(v)<end(v)<end(u)$且$u.level=v.level-1$,其中level表示结点位于文档树中的层次.

XML文档树结点编码完成后,接下来的工作是对文档树结点建立索引.其索引方法是:索引表中的索引项的数据结构为node(num, elementname, begin, end, level, ancestornum),其中num表示层次遍历文档树所产生的序号;elementname表示结点标签名;begin表示结点的起始编码;end表示结点的结束编码;level表示结点所在文档树中的层次;ancestornum记录结点的所有祖先结点的位置,它在多分支查询算法depthjoin中起着很重要的作用,通过它可以快速地找到相应结点到根结点的路径上的任一祖先结点.对应的索引表如表1所示.例如,图2中TITLE结点的索引结构为(1, TITLE, 2, 4, 2, 0).

```
<PLAY>
  <TITLE>A1</TITLE>
  <ACT>
  <TITLE>A2</TITLE>
  <SCENE>
    <SPEECH>
    <SPEAKER>A3
    </SPEAKER>
    <LINE>A4</LINE>
    </SPEECH>
    <STAGEDIR>A5
    </STAGEDIR>
  </SCENE>
  </ACT>
  <fn>
    <p>A6</p>
    <p>A7</p>
    <p>A8</p>
    <p>A9</p>
  </fn>
  <ACT>
    <TITLE>A10</TITLE>
    <SCENE>
    <SPEECH>
      <SPEAKER>
    <NAME>A11</NAME>
      </SPEAKER>
      <PARAGRAPH><LINE>
A12</LINE></PARAGRAPH>
      <PARAGRAPH><LINE>
A13</LINE></PARAGRAPH>
    </SPEECH>
     <STAGEDIR>A14</
STAGEDIR>
    </SCENE>
  </ACT>
  </PLAY>
```

图1　XML文档

Fig 1　A XML document

表1　XML索引表

Table 1　XML indexes

num	elementname	begin	end	level	ancestornum
0	PLAY	1	68	1	
1	TITLE	2	4	2	0
2	ACT	5	26	2	0
3	fn	27	40	2	0
4	ACT	41	67	2	0
5	A1	3	3	3	1, 0
6	TITLE	6	8	3	2, 0
7	SCENE	9	25	3	2, 0
8	p	28	30	3	3, 0
9	p	31	33	3	3, 0
10	p	34	36	3	3, 0
…	…	…	…	…	…
30	A14	64	64	5	23, 13, 4, 0
31	NAME	12	14	6	24, 15, 7, 2, 0
32	LINE	17	19	6	25, 15, 7, 2, 0
33	NAME	48	50	6	27, 22, 13, 4, 0
34	LINE	53	55	6	28, 22, 13, 4, 0
35	LINE	58	60	6	29, 22, 13, 4, 0
36	A3	13	13	7	31, 24, 15, 7, 2, 0
37	A4	18	18	7	32, 25, 15, 7, 2, 0
38	A11	49	49	7	33, 27, 22, 13, 4, 0
39	A12	54	54	7	34, 28, 22, 13, 4, 0
40	A13	59	59	7	35, 29, 22, 13, 4, 0

2 多分支路径查询算法 depthjoin

2.1 有关概念

定义 **1** 单路径: 每个结点有且仅有一个子结点, 并且除第一个结点外, 每个结点有且仅有一个父结点. 如路径 a//b//c/d/e即为单路径. 其中“ //”表示两结点之间的关系为祖先 后裔关系, “ /”表示两结点之间的关系为父子关系. 其所对应的查询图如图 3(a)所示.

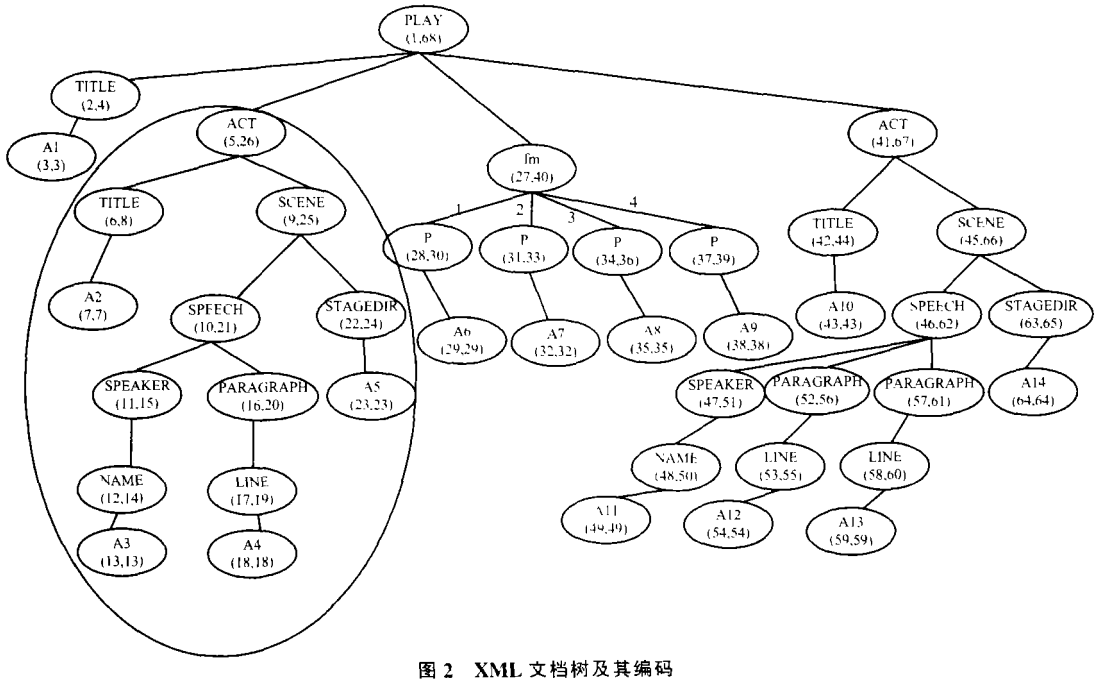

图 2 XML 文档树及其编码

Fig.2 XML tree and its encoding

定义 **2** 多分支路径: 有且仅有一个根节点, 每个结点(除根结点)有且仅有一个父结点, 每个结点可以有多个子结点. 如路径 a[/b] [//c[/d] [/e]], 其所对应的查询图如图 3(b)所示.

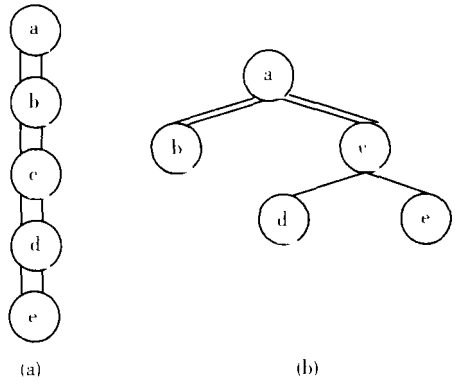

图 3 查询路径种类

Fig.3 Category of query paths

2.2 算法思想

算法 depthjoin的基本思想是:

(1) 对 XML文档进行扫描, 编码并建立 XML文档树.

(2) 层次遍历 XML文档树, 建立 XML索引表.

(3) 根据查询语句建立查询树, 并从索引表获得查询树中每个结点所对应的索引信息. 若查询树中的结点在索引表中无索引信息, 则查找失败.

(4) 按后根遍历查询树的顺序进行 XML查询匹配. 其中, 运用 pathstack算法[9] 在 XML文档中查找单路径; 在处理查询树中多分支结点 P 时, 利用索引信息判断结点 P 的所有孩子在文档树中是否存在共同的祖先 P, 若存在, 则保存祖先为 P 的分支路径, 继续下一个结点的匹配, 否则查找失败.

(5) 查询树遍历结束后得出查询树在 XML文档中的匹配结果.

2.3 算法描述

算法 depthjoin的描述如下:

```
void depthjoin( tree*  root)        /查询树采用孩子-兄弟链表存储, root为根指针
{     travers_ index( root);        /遍历查询树, 将索引表的信息记入查询树相应结点中
      createstack( s1);             /建立栈 s1, 用于非递归后根遍历查询树, 即中序遍历孩子-兄弟链表
```

```
        createstack(s2);                    //建立栈 s2,用于存储单分支结点
        p= root;
        while (p || ! StackEmpty(s1))
        {   if (p) {push(s1, p); p= p-> lchild;}        //遍历第一棵子树
            else {
            pop(s1, p);
            if (p所指结点为叶结点或只有一个孩子)        //p所指结点为单分支结点
                push(s2, p);                            //保存结点信息
            else                                        //p所指结点为多分支结点
        brachpath= indexquery (p, singlepath);          //进行多分支结点的匹配;
//利用索引信息判断结点 p的所有孩子在文档树中是否存在共同的祖先 p,若存在,则保存祖先为 p的分支路径,继续查找,否则查找失败.
        if (p的父结点是多分支结点 && ! StackEmpty(s2)) singlepath= pathstack(s2);
        //运用 pathstack算法在文档树中查询栈 s2中单路径结点,并清空栈 s2;
      p= p-> rchild;     //遍历下一棵兄弟子树
    } //endif
  } //endwhile
}
```

3 实验及结果分析

为了验证算法 depthjoin的效率,利用 VC++实现了算法 depthjoin.实验环境为: PentiumⅣ 1.60G CPU, 256MB内存, Windows 2000.实验数据为莎士比亚 XML数据集[10].为了便于算法的性能比较,将多分支查询树拆分成由根结点到叶子结点的若干单路径,使用 pathstack算法对单路径分别进行查询,最后将单路径的查询结果进行连接操作得到最终的分支查询结果,本文将这种算法称为 pathstackjoin算法.在实验中,测试了莎士比亚 XML数据集的 10个分支查询语句,查询语句如表 2所示.对于这些多分支查询,分别使用算法 depthjoin和算法 pathstackjoin进行查询.它们的查询效率对比如图 4所示.实验表明算法 depthjoin的性能要优于算法 pathstackjoin.可以看出,当查询路径分支结点较多、查询路径长度较长且查询结点的索引项较多时,如查询语句 P2、P3、P4、P6、P10,算法 depthjoin的效率具有明显的优势.

表 2 Shakespeare数据集多分支查询用例

Table 2 Multiple branch query examples of shakespeare data set

路径 ID	查询表达式
P1	SCENE[//LINE][/STAGEDIR]
P2	PLAY[//p][/PERSONAE[//PERSONA][/PGROUP[//PERSONA][/GRPDESCR]]]
P3	PLAY[//PERSONA][/ACT[//TITLE][//SPEECH[/SPEAKER][/LINE]]]
P4	SCENE[//SPEECH[//SPEAKER][//LINE]][//SPEECH[//LINE][//STAGEDIR]]
P5	ACT[//SPEECH][//LINE]
P6	PLAY[//ACT[//SPEECH][//LINE]][//ACT[//SPEECH][//LINE]]
P7	ACT[//SPEECH][/LINE/Look, where they come:]
P8	PGROUP[//PERSONA][/GRPDESCR]
P9	PERSONAE[//TITLE/Dramatis Personae][//PGROUP[//PERSONA][/PERSONA]]
P10	ACT[/SCENE[//SPEECH][//STAGEDIR]][//SPEECH[//SPEAKER][//LINE]]

4 结束语

对 XML文档实现多分支查询,一般方法是将多分支路径拆分成若干个单路径,对单路径分别进行查询,然后再对查询结果进行连接操作,这样的算法效率低.而本文提出的 depthjoin算法为实现 XML多分支快速查询提供一种有效方法.该算法在对查询树进行匹配的过程中,不需要对多分支路径进行拆分且避免了单路径的连接操作,它充分利用索引技术来判定分支结点的子结点是否具有共同的祖先或父结点,从而大大提高了查询效率.

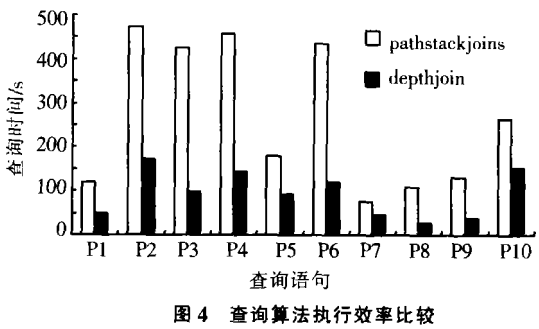

图 4 查询算法执行效率比较

Fig.4 Comparison of query algorithms efficiency

[参考文献] (References)

[1] Kaushik R, Shenoy P, Bohannon P, et al. Exploiting local similarity for efficient indexing of paths in graph structured data [C] // 10th International Conference on Database Theory. California: San Jose, 2002: 129-140.

[2] Chen Q, Lim A, Ong K W. D(k)-index: an adaptive structural summary for graph-structured data[C] // Proc of the 2003 ACM SIGMOD Intl Conf on Management of Data. California: San Diego, 2003: 134-144.

[3] Chung C, Min J, Shim K. APEX: an adaptive path index for XML data[C] // Proc of the 2002 ACM SIGMOD Intl Conf on Management of Data. Wisconsin: Madison, 2002: 121-132.

[4] Milo T, Suciu D. Index structures for path expressions[C] // 7th International Conference on Database Theory. Israel: Jerusalem, 1999: 277-255.

[5] Li Quanzhong, Bongki Moon. Indexing and querying XML data for regular path expressions[C] // Proceedings of the 27th VLDB Conference. Italy: Roma, 2001: 361-370.

[6] Roy Goldman, Jennifer Widom. DataGuide: enabling query formulation and optimization in semistructured databases[C] // Proceedings of the 23th International Conference on Very Large Data Bases. Athens: Pages, 1997: 436-445.

[7] Jagadish H V S, Koudas N. Structural joins: a primitive for efficient XML query pattern matching[C] // Hiong Ngu A H. Proceedings of the 18th IEEE ICDE International Conference on Data Engineering. California, 2002: 141-152.

[8] Chien S Y, Vagena Z, Zhang Donghui, et al. Efficient structural joins on indexed XML document[C] // Papadias D. Proceedings of the 28th VLDB International Conference on Very Large Database. China: Hong Kong, 2002: 263-274.

[9] Bruno N, Koudas N, Srivastava D. Holistic twig Joins: Optimal XML pattern matching[C] // Franklin M J. Proceedings of the 21th ACM SIGMOD International Conference on Management of Data. Wisconsin: Madison, 2002: 310-321.

[10] Shakespeare XML data sets[DB/OL]. ftp://sunsite.unc.edu/pub/sun-info/standards/XML/egs/.

[责任编辑: 严海琳]