文章编号: 1005-3026(2005)06-0535-03

XML路径表达式中公共子查询的优化技术

韩东红, 王国仁, 乔百友
(东北大学 信息科学与工程学院, 辽宁 沈阳 110004)

摘 要: 研究了XML路径表达式的相关查询算法,提出了基于标注后缀树的XML路径表达式公共子查询的优化算法,通过冗余消除技术来提高路径表达式的查询效率.在REOA测试集上,通过对设计的查询进行测试,分析了基于标注后缀树的XML路径表达式公共子查询的优化算法的性能.实验结果表明,基于标注后缀树的XML路径表达式冗余消除技术可以极大地提高路径表达式的查询效率.

关 键 词: XML路径表达式;公共子查询;查询优化;标注后缀树;冗余消除技术
中图分类号: TP 311.13 文献标识码: A

1 问题的提出

对于XML路径表达式查询外延连接算法Extent-Join[1]以及其他基于连接的查询算法,例如文献[2,3]在文中所采用的方案来说,在处理某些复杂查询时,如果不采取适当措施的话,会造成冗余计算问题.图1给出了路径表达式查询"/(((A/B)/C)/D)/E/(((A/B)/C)/D)/F"相对应的逻辑查询树形状,为了叙述方便,对非叶子进行了编号.从图1可以清楚地知道,操作1与5,2与6以及3与7是两两完全相同的,这就造成至少有3次冗余连接操作,而其实这些操作是完全可以避免的.下面是解决这一问题的通常途径: ① 将可能重复利用的已经完成的操作结果暂存; ④在进行新的操作前查看本操作是否已经执行过,如果是则取得暂存结果,否则执行操作.

在图1中,操作1= 操作5= A ∞B,操作2= 操作6= (A ∞B) ∞C,操作3= 操作7= ((A ∞B) ∞C) ∞D,由此可见通过使用暂存结果集合可避免3次冗余连接操作.

然而,对于大多数查询来说发现冗余操作不是显而易见的事情,因为用户书写查询时是不会有这方面考虑的.实际上,对于图1中的查询,用户更有可能写成"/A/B/C/D/E/A/B/C/D/F"形式.对于这样的查询,其查询树是一棵完全左深树,树上没有任何相同功能的非叶子节点.解决的方法就是在查询优化过程中将完全左深树转换为图1的形式.那么现在的问题在于:如何转换查询树形状?

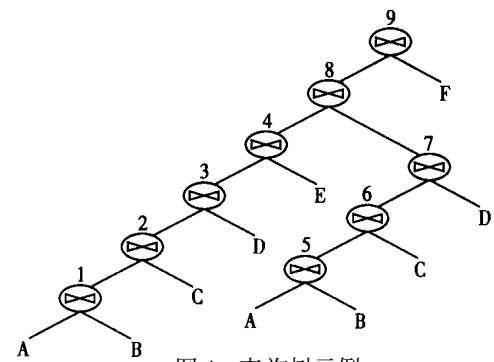

图1 查询树示例
Fig. 1 Example of query tree

这个查询优化工作可以是在查询转换后进行的查询树型转换,也可以是在查询转换之前进行的用户路径表达式查询形式上的变化.分析路径表达式查询 a = "/(((A/B)/C)/D)/E/(((A/B)/C)/D)/F" 与 b = "/A/B/C/D/E/A/B/C/D/F",发现查询 a 与查询 b 的区别在于 a 把可能重复的操作使用优先级操作符(( ))"括了起来",从而在查询转换过程之前就达到了目的.如果将路径查询看作是字符串的话,那么这个查询优化问题就转换为快速有效地寻找字符串重复子串问题.在这个问题上,作为线性时间字符串子串查找方案的后缀树是一个理想的解决工具.

收稿日期: 2004-09-17
基金项目: 国家自然科学基金资助项目(60273039).
作者简介: 韩东红(1968-),女,河北平山人,东北大学讲师,博士研究生; 王国仁(1966-),男,湖北崇阳人,东北大学教授,博士生导师.

2 公共子查询优化算法

后缀树是一种非常重要的数据结构, 可以应用到诸多领域, 例如分子生物学、数据处理、字符串处理、信息检索、信号处理等. 对后缀树的研究可以追溯到 20 世纪 60 年代末, Morrison 在 1968 年首次隐含地论述了后缀树的概念, 称为 Patricia 树, Weiner 在 1973 年首次明确提出了后缀树的概念, 称为紧凑型b-tree[4]. 由于在后缀树上要表示一个字符串的所有后缀以便发现满足条件的子串和序列模式, 因此对空间的要求非常高. 为解决这一问题, 人们又提出了一些后缀树的变种, 例如, 后缀数组[5]、后缀二叉查找树[5]、惰性后缀树[6]、后缀 Cactus(后缀树和后缀数组的结合)[7]. 为了将多个基因序列能够存储在一个后缀树中, 一种通用后缀树-GST 树[8]被提出. 这些后缀树及其变种数据结构所需要的内存开销都非常大, 因此人们对后缀树的压缩存储进行了研究, 以减少或缓解内存大小的约束[9,10].

图 2 给出了字符串"/A/B/C/D/E/A/B/C/D/F"所对应的路径后缀树片断. 由于后缀树在本文中用于路径表达式查询优化, 因此以路径连接符('/'、'//')或者操作符('|'、'('、')'、'*'、'+')开头与结尾的字符串被舍弃. 得到的后缀有{"F", "E/A/B/C/D/F", "D/F", "D/E/A/B/C/D/F/", "C/D/F", "C/D/E/A/B/C/D/F", "B/C/D/F", "B/C/D/E/A/B/C/D/F", "A/B/C/D/F", "A/B/C/D/E/A/B/C/D/F"}.

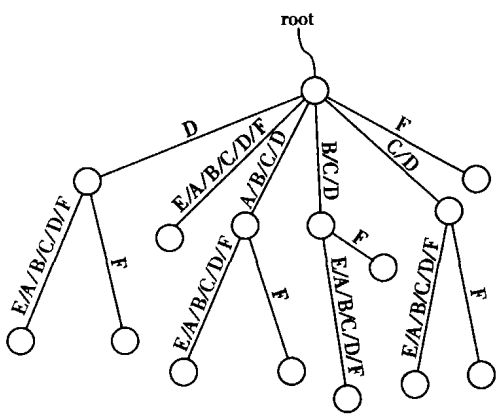

图 2 路径后缀树示例

Fig. 2 Example of path suffix tree

路径表达式公共子查询优化的关键在于从路径后缀树取得重复子路径的过程. 由于后缀树设计上是用于判定某个字符串是否为另一个字符串的子串, 没有子串重复次数设定. 因此在后缀树上加以改进, 将后缀树的节点上标注子串在原字符串中出现的次数, 称为标注后缀树. 加注重复次数的方法非常简单, 只是在后缀树的建树算法上稍稍改动即可. 图 3 给出了图 2 所示的后缀树相对应的标注后缀树.

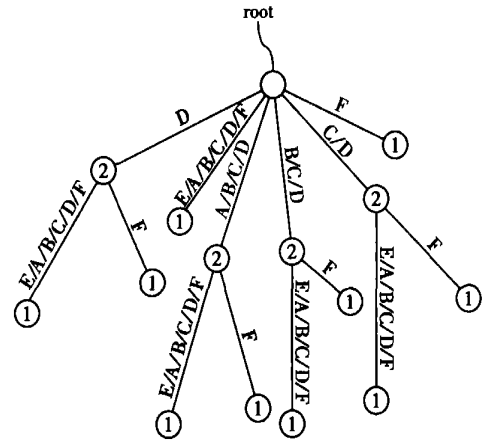

图 3 标注路径后缀树示例

Fig. 3 Example of marked path suffix tree

通过对标注后缀树的分析, 容易得到查询路径字符串中重复的子串. 例如对于图 2 所示标注后缀树, 重复的子串以及重复次数如表 1. 由于长度为 1 的子路径在优化中没有用处, 表中只列出长度大于 1 的重复子串.

表 1 重复子查询字符串

Table 1 Duplicate path strings

子查询字符串	重复次数
A/B/C/D	2
A/B/C	2
A/B	2
B/C/D	2
B/C	2
C/D	2

对于查询路径字符串的标注原则是: 在查询字符串对应标注后缀树得到的重复子串中以长度从长到短, 位置由左至右的顺序取子串并进行加注括号(优先级)操作. 因此对于表 1 中的子串处理顺序为"A/B/C/D" → "A/B/C" → "B/C/D" → "A/B" → "B/C" → "C/D". 这样必然有的子串由于其他子串先被处理而无法标注将被放弃, 例如在"A/B/C"被加注后字符串"/A/B/C/D/E/A/B/C/D/F"成为"/((A/B/C)/D)/E/((A/B/C)/D)/F", "B/C/D"将因无法继续标注而被舍弃. 经过整个标注过程的查询字符串为"/(((A/B)/C)/D)/E/(((A/B)/C)/D)/F". 值得注意的是, 处理子串的顺序是由查询转换算法决定的, 如果查询转换算法的基本方法是基于右深树原则, 那么子串处理顺序就将是由右至左. 至此, 可以得到 XML 路径表达式公共子查询的优化过程, 可用算法 1 表述.

算法 1 冗余消除优化算法 REOA

(PathStrings)

输入: 路径查询字符串 s

输出: 标注路径查询字符串 s'

(1) 对于路径查询字符串 s 构建路径后缀树;

(2) 取得重复子路径字符串 str_i;

(3) 使用子路径字符串 str_i 标准查询 s, 得到标准后的查询路径 s'.

3 性能评价

对于复杂查询来说, 有可能隐藏重复操作, 尤其对于基于连接的路径表达式查询处理技术, 重复连接操作的代价是巨大的, 也是可以避免的. 图 4 给出了冗余消除技术在三个查询上的性能. 为了有效测试 REOA 技术的性能, 专门设计了三个查询, 如表 2 所示. 从图中可以看出 REOA 的性能是显著的, 尤其在重复子查询比较多的时候性能可以提高若干倍, 例如 Q3 经过优化后的效率比原查询提高了 3 个数量级.

表 2　冗余消除优化算法测试查询

Fig. 2　Testing queries of REOA

编号	原查询	优化查询	重复子查询
Q1	a/b/d/e/g/k/b/d/e/g/j/c/f/h	a/(b/d/e/g)/k/(b/d/e/g)/j/c/f/h	b/d/e/g
Q2	a/d/e/g/k/b/d/e/g/k	a/(d/e/g/k)/b/(d/e/g/k)	d/e/g/k
Q3	a/b/c/f/g/k/b/c/f/g/k/b/c/f/h	a/((b/c/f)/g/k)/(b/c/f/g/k)/(b/c/f)/h	b/c/f/g/k, b, /c/f

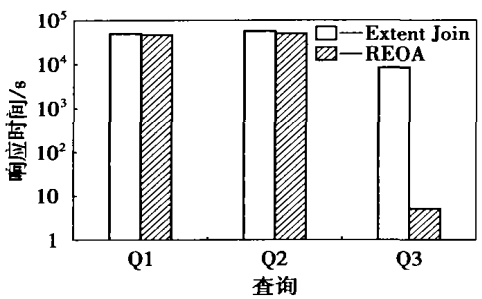

图 4　性能评价

Fig. 4　Performance evaluation

4 结　论

本文研究了路径表达式查询处理中的公共子查询优化技术问题, 并提出和设计了一个基于标注后缀树的查询优化算法, 然后通过实验验证了该查询优化算法比原查询效率提高若干倍, 证明了它的有效性.

参考文献:

[1] Wang G R, Lv J H, Sun B, *et al*. RPE Query processing and optimization techniques for XML databases[J]. *Journal of Computer Science and Technology*, 2003, 19(2): 224-238.

[2] Zhang C, Naughton J F, de Witt D J, *et al*. On supporting containment queries in relational database management systems[A]. *Proceedings of* 2001 *SIGMOD Conference* [C]. Santa Barbara, 2001. 425-436.

[3] Li Q, Moon B. Indexing and querying XML data for regular path expressions[A]. *Proceedings of the 27th VLDB Conference*[C]. Roma: Morgan Kaufmann Publishers, 2001. 361-370.

[4] Giegerich R, Kurtz S, Stoye J. Efficient implementation of lazy suffix trees[A]. *Proceedings of 3rd Workshop on Algorithm Engineering* [C]. London: Springer-Verlag, 1999. 30-42.

[5] Irving R W, Love L. The suffix binary search tree and suffix AVL tree[J]. *Journal of Discrete Algorithms*, 2003, 1(5): 387-480.

[6] Bieganski P, Riedl J, Carlis J, *et al*. Generalized suffix trees for biological sequence data: applications and implementation [R]. Minneapolis: University of Minnesota, 1999. 1120-1156.

[7] Karkkainen J. Suffix cactus: a cross between suffix tree and suffix array[A]. *Proceedings of the Annual Symposium on Combinatorial Pattern Matching (CPM'95)*[C]. Helsinki, 1995. 191-204.

[8] Kurtz K. Reducing the space requirement of suffix trees[R]. Brussel: University of Helsinki, 1999. 1149-1171.

[9] Gross R, Vitter J S. Compressed suffix arrays and suffix trees with applications to text indexing and string matching [A]. *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing*[C]. Portland, 2000. 397-406.

[10] Larsson N J. Extended application of suffix trees to data compression[A]. *Proceedings of the 6th Data Compression Conference (DCC'96)*[C]. Snowbird: IEEE Computer Society Press, 1996. 190-199.

Optimizing Common Sub-Queries in XML Data for Regular Path Expressions

HAN Dong-hong, WANG Guo-ren, QIAO Bai-you

(School of Information Science & Engineering, Northeastern University, Shenyang 110004, China. Correspondent: HAN Dong-hong, E-mail: hdhcome@163.com)

Abstract: Focusing on the query algorithm of XML data for path expressions and based on the marked suffix tree (MST), an optimizing algorithm is proposed for the common sub-queries in XML data for regular path expressions to improve the querying efficiency of path expressions through the redundancy eliminating technique. After a test done on the benchmark REOA, a performance analysis is made to the optimizing algorithm for the common sub-queries on MST basis. The result shows that the redundancy eliminating technique based on MST and used in XML data for path expressions will greatly improve the querying efficiency.

Key words: XML data regular path expressions; common sub-queries; query optimization; marked suffix tree (MST); redundancy eliminating technique

(Received September 17, 2004)