维普资讯 http://www.cqvip.com



# XML 文档顺序的维护*)

黄 芳 孙建伶
(浙江大学计算机学院人工智能研究所 杭州310027)

**摘 要** 为提高查询和更新的效率,XML 文档中的对象必须拥有顺序标识(OID)。本文在 Numbering sheme[1]的基础上提出了基于模式的顺序维护算法。新算法包括 OID 区间预留算法和 OID 维护算法两部分。在文档初始化时,前者基于模式和统计信息对未来的插入操作进行 OID 区间预留,后者则在前者的基础上将插入操作时的全局调整化解为局部调整。新算法可以有效降低 XML 文档顺序维护的代价。

**关键词** XML,预留区间,节点长度,局部调整

## Maintaining Order in an XML Document

HUANG Fang SUN Jian-Ling
(Institute of Artificial Intelligence, College of Computer Science, Zhejiang University, Hangzhou 310027)

**Abstract** Sequential order should be maintained in an XML document to improve the efficiency of query and update operations upon it. This paper proposes a Schema-based order maintaining algorithm after Numbering scheme[1]. The new algorithm mainly includes two parts. The first is OID reserving, which implements OID space reservation based on the Schema and statistic information of a sample during initialization. The second one is OID maintaining, which degrades globe reordering to local reordering based on the first part. The new algorithm reduces maintenance overhead effectively.

**Keywords** XML, Reserved space, Node's length, Local reordering

## 1 引言

XQuery[2]的实现技术[3,4]是当前的一个研究热点。XML 文档查询处理系统需要对 XML 文档中的对象进行标识,为此引入了对象标识(OID),XML 中的 OID 是 OODB 中 OID 的扩充[1]。扩充的主要原因是 XML 文档中对象的顺序蕴涵语义。以图书管理系统[5]为例,一本书通常有数位按序排列的作者,作者的顺序蕴涵了"作者的重要性"这一重要语义,采用 OODB 中的 OID 会造成该语义的丢失。同时,大部分查询算法均要求快速标识节点之间的 ancestor-descendant 关系[1],而且 XQuery 中有许多基于语义的更新操作如 INSERT NewNode [ BEFORE|AFTER RefNode]等。因此,XML 文档中的对象必须拥有顺序标识。通常,XML 文档中的对象成树状排列。最早为树状结构文档提供顺序标识的算法是前序算法[6],但是该算法的平均代价很大($O(N/2)$)。随后,Paul F. Dietz 对前序算法进行了改进并提出了 Dietz's scheme,通过对树中的节点分层,将平均代价降为 $O(I+C\log m)$[7]。由于 Dietz's scheme 是针对2-3 tree 设计的,所以将其应用到 XML 文档中时缺乏灵活性,因此,Quanzhong Li 等[1]提出了 Numbering scheme,该算法用一对整数(order, size)标识 XML 文档中的对象,并首次采用了预留的思想。但是,Quanzhong Li 等对算法本身未进行深入分析,尤其是对如何实现预留未给出解释。此外,Numbering scheme 仅仅是作者为了建立高效的索引机制而提出的,对于 XML 的其他重要应用如更新等,并未给出解决方案。本文在 Numbering scheme 的基础上提出了基于模式的顺序维护算法。新算法对 Numbering scheme 进行了全方位的扩充,首先,新算法以模式作为 OID 区间的预留的基础,有效避免了 Numbering scheme 的盲目性;其次即实例文档统计信息的利用,有利于充分采集现实世界的信息,给出更合理的预留方案。文中重点给出了 OID 区间预留算法和 OID 维护算法。前者基于模式和统计信息实现文档初始化时的 OID 区间预留,后者则在前者的基础上将插入操作时的全局调整有效化解为局部调整。我们以应用最广泛的 XML 顺序标识算法--前序算法作为新算法的比较对象。

本文共分4部分。第1部分是引言;第2部分是基本概念;第3部分是基于模式的顺序维护算法及其分析;最后是结论。

## 2 基本概念

### 2.1 相关定义

**定义1**(节点长度) 在 XML 文档中插入一个节点所需的 OID 区间的长度,称为该节点的节点长度。通常,OID 区间是一个整数闭区间。

**定义2**(子孙节点) 给定一个节点 $E$ 和节点集合 $U$,如果 $U$ 中每个节点的父亲或者祖先是 $E$,那么集合 $U$ 中的节点均是节点 $E$ 的子孙节点。以图1为例,元素 Bib 的子孙节点包括除 Bib 之外的所有节点(不含注释(# text))。

**定义3**(元素节点和关系节点) 根据 W3C 的规定,由 ELEMENT 等关键字指定的节点为元素节点;而由 SEQUENCE、CHOICE 等关键字指定的节点为关系节点。以图1为例,椭圆形节点为元素节点,八角形节点为关系节点,长方形节点为注释。

*)基金项目:航天工业总公司国防预防研基金资助项目(2000-002CAD)。

维普资讯 http://www.cqvip.com

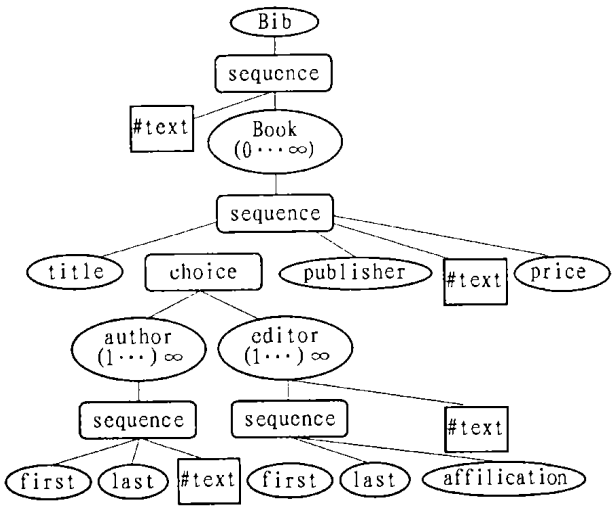


图1 元素 Bib 及其子孙节点

**定义4**(文档模式的重叠深度) 文档模式[8]的重叠深度指文档模式中"unbounded"关键字重叠出现的次数。通常,文档模式的重叠深度小于等于文档模式树的深度。以图1为例,文档模式树的深度为8,而重叠深度为2。

### 2.2 节点长度计算公式

给定一个节点 $E$,它的长度计算公式为

$$Len(E)=\begin{cases}\sum_{i=1}^{r}(Len(V_i)\times T_i)+1 & E\text{ 是元素节点}\\ \mathrm{Max}_{i=1,2,\cdots,r}Len(V_i) & E\text{ 是 CHOICE 定义的关系节点}\\ \sum_{i=1}^{r}(Len(V_i)\times T_i) & E\text{ 是其他关系节点}\end{cases}\quad(1)$$

其中 $r$ 为 $E$ 的子节点个数;$V_i$ 为 $E$ 的第 $i$ 个子节点;$T_i$ 为 $V_i$ 的出现次数约束。以图1中的元素节点 *editor* 为例,它的长度计算如下:

$$\begin{aligned}Len(editor)&=len(sequence)\times 1+1\\&=(Len(first)\times 1+Len(last)\times 1+Len(affilication)\times 1)\times 1+1\\&=1+1+1+1=4\end{aligned}$$

## 3 基于模式的顺序维护算法

基于模式的顺序维护算法包括 OID 区间预留算法和 OID 维护算法两部分,分别在3.1节和3.2节介绍。3.3节是算法比较和分析。

### 3.1 OID 区间预留算法

文档模式可以定义元素的出现次数约束[8]。以图1中的图书管理系统为例,带有{0…∞}标识的节点如 *book*、*author* 等在其父节点下最多允许出现"*unbounded*"次;边框为虚线的节点如 *book* 等在其父节点下最少允许出现零次;没有特殊说明的节点如 *price*、*first* 等在其父节点下必须且只能出现一次。

根据文档模式,OID 区间预留规则设计如下:首先,对于允许出现 $C(C>=1)$ 次的节点,直接预留出 $C$ 倍该节点长度的 OID 区间;其次,对于允许出现"*unbounded*"次的节点,借助于实例文档的统计信息计算该类节点的预留区间。统计信息的采用基于一个惯性假设---实例中出现次数多的元素类型未来出现的频率高。该假设的采用有利于充分采集现实世界的信息。以节点 *book* 和 *author* 为例,虽然它们最大允许出现次数均为"*unbounded*",但是一本书的作者最多几十个,而一个图书馆的藏书却可以有上百万册,因此将这两个节点区别对待更为合理。具体做法如下:遍历实例文档,求取 *book* 在 *Bib* 中、*author* 在 *book* 中的最大出现次数,分别记为 $Max_{bib-book}$ 和 $Max_{book-author}$;引入公共系数变量 $x$,将两个"*unbounded*"约束分别转化为 $Max_{bib-book}\times x$ 和 $Max_{book-author}\times x$。

基于预留规则,OID 区间预留算法设计如下:

```
Algorithm OIDReserving (Schema,Document,I)
Begin
    for Document 中的每一对父子节点(P,E)
        If E 在 P 下允许出现"unbounded"次 then
            求取统计值 Max_{P-E}
    for Schema 中的每个节点 E
        根据公式(1)求算该节点的长度 Len(E,x);
        令 Schema 根节点的长度等于 Len(I),求该方程的近似解,得 x;
    for Schema 中的每个节点 E
        将 x 代入 Len(E,x),得 E 的节点长度;
End
```

在该算法中,$I$ 为一个 OID 区间,$Len(I)$ 为区间 $I$ 的长度。在文档初始化时,$I$ 通常是计算机支持的最大正整数区间 $[0,2^{32}]$。由于一对父子节点 $(P,E)$ 的统计值 $Max_{P-E}$ 可能为零,从而导致方程的畸形,因此,区间预留算法允许用户设定一个缺省的倍率 $Max$。当 $E$ 在 $p$ 下的允许出现次数为"*unbounded*"且 $Max_{P-E}$ 小于 $Max$ 时,相应的"*unbounded*"约束转化为 $Max\times x$。该算法的主要代价是求解如下 $n$ 次方程的近似解

$$a_1x^{n-1}+a_2x^{n-1}+\cdots+a_nx+a_{n+1}=Len(I)\quad(2)$$

根据算法描述,方程(2)的系数满足 $a_1>a_2>\cdots>a_n>Max$,因此,方程有唯一正根 $x\in[0,\sqrt[n]{\frac{Len(I)-a_{n+1}}{a_1}}]$,采用二分法[10]即可求得方程(2)的近似解。对重叠深度为 $n$ 的文档模式,算法的复杂度为 $O(n^2)$(一般文档模式的重叠深度都不大,比如,图1的重叠深度为2)。

### 3.2 OID 维护算法

我们首先介绍前序算法的 OID 维护。采用前序算法时,对于规模为 $D$ 的 *Document*,新插入一个节点的平均代价 $Cost_{preserial}$ 为 $D/2$。

```
Algorithm PreOrder (Document,NewNode)
  Begin
      for Document 中的每个节点 E
          if E 在新插入的节点 NewNode 之后 then
              更新节点 E 的 OID;
  End
```

为了降低文档维护的代价,我们的 OID 维护算法则将发生在整个文档范围内的调整尽可能化解为局部调整,3.1节的 OID 区间预留算法为它做了充分的铺垫。在删除操作时,OID 的回收不需要任何代价[1],所以只需分析插入操作时的 OID 维护。由于 INSERT *NewNode* AFTER *RefNode* 和 INSERT *NewNode* BEFORE *RefNode* 没有本质区别,因此,我们只给出前者的 OID 维护算法。

```
Algorithm OIDMaintaining (Schema,Document NewNode,RefNode)
Begin
    获取 NewNode 及其父节点 ParentNode 的元素模式,分别记为 E 和 P;
    if E 在 P 下的最大允许出现次数≤1 then
        不需要任何调整;
    else
        if RefNode 前 OID 空白区间的长度≥NewNode 的节点长度 Len(E) then
            不需要任何调整;
        else
            计算 ParentNode 下为 E 类型节点预留的 OID 区间 I_reserve;
            遍历 I_reserve,寻找其中最长的空白区间 I_bland=[P_left,P_right];
if Len(I_blank)≥Len(E) then
  if I_blank在 RefNode 之左 then
```

维普资讯 http://www.cqvip.com

```
    将 P_right(含)和 RefNode (不含)之间的节点的 OID 更新;
else
    将 RefNode (含)和 P_left(含)之间的节点的 OID 更新;
else
    if E 在 p 下的最大允许出现次数为 C(C>1) then
      该插入操作违反 Schema;
    else
      执行循环 Loop (Schema,NewNode,NewNode);
End

Algorithm Loop(Schema,Node,NewNode)
Begin
    If Node 已经是 Schema 的根节点 then
      实例树已经饱和,返回;
    else
      以 Node 的父节点 ParentNode 为根节点,对 ParentNode 占
        据的区间重新分配;
      if 重新分配后可以为 NewNode 赋予 OID then
        更新 ParentNode 所有子孙节点的 OID 并返回;
      else
        执行循环 Loop (Schema,ParentNode,NewNode);
End
```

### 3.3 算法比较和分析

我们首先分析 OID 区间预留算法。与前序算法相比,该部分是额外代价。根据3.1节可知,该算法的核心是求解一个 $n$ 次方程的近似解,$n$ 通常很小,并且由于方程系数的特殊性使得方程求解的代价很小,与 OID 更新操作相比,该代价可以忽略。因此,我们只需重点分析 OID 维护算法。

根据算法描述,插入操作时一个局部 OID 区间的维护过程等价于 HB($k$)(height-balanced $k$-tree)[9]二叉搜索树的维护,这里的 $k=\max\{floor(\log x),1\}$($floor$ 为取整函数,以下同)。因此,若记 $F(C,Len(E))$和 $G(\tilde{C},\text{Document})$分别是出现 $C(C>1)$次和 *unbounded* 次以下的代价表达式,则有

$$F(C,Len(E))=O(\log C)\times Len(E)$$

$$\begin{aligned}G(\tilde{C},\text{Document})&=(1-\theta_i)\times[O(\log\tilde{C})\times Len(E)]+(1-\theta_{m+1})\theta_m\times Len(E_{m+1})\\&\quad+(1-\theta_{m+2})\theta_{m+1}\theta_m\times Len(E_{m+2})+\cdots\\&\quad+(1-\theta_l)\theta_{l-1}\cdots\theta_m\times Len(E_l)\\&=O(\log\tilde{C})\times Len(E)+\sum_{i=m+1}^{l}[(1-\theta_i)\prod_{j=m}^{i-1}\theta_j]Len(E_i)\end{aligned}$$

其中 $\tilde{C}$ 为"*unbounded*"约束的节点的出现次数的计算值;$m$ 为 NewNode 节点在 Document 树中的深度,$l$ 为 Document 树的深度;$\theta_i$ 为在第 $i$ 层进行局部调整失败的概率,$\theta_i(i=m,m+1,\cdots,l)$之间互相独立;$E_i(i=m+1,\cdots,l)$分别为 NewNode 的父亲节点、祖父节点,…,Document 的根节点,显然,$Len(E_m)=Len(E)$,$Len(E_l)=D$。若记 $\alpha$ 和 $\beta$ 分别是 NewNode 允许出现 $C(C>1)$次和 *unbounded* 次的节点概率,则在规模为 $D$ 的 Document 中插入一个新节点的平均代价($Cost_{OIDMaintaining}$)为

$$\begin{aligned}Cost_{OIDMaintaining}&=\alpha[O(\log C)\times Len(E)]+\beta\{O(\log\tilde{C})\times Len(E)+\sum_{i=m+1}^{l}[(1-\theta_i)\prod_{j=m}^{i-1}\theta_j]Len(E_i)\}\\&=[O(\log C)+O(\log\tilde{C})]Len(E)+\beta\sum_{i=m+1}^{l}[(1-\theta_i)\prod_{j=m}^{i-1}\theta_j]Len(E_i)\end{aligned}$$

由于实例初始化时对未来的插入操作进行了 OID 区间预留,因此,小规模的插入几乎不需要任何调整,而且对于规模很大的插入,局部调整失败的概率也很小,即:$\theta_i(i=m,m+1,\cdots,l)$很小。以图1为例,假定现有图书$10^6$册,每本书均有50个 *editor*,采用 *OID* 区间预留算法进行实例初始化。不妨假定缺省倍率 $Max=50$,初始化区间 $I=[1,2^{32}]$,构造方程求近似解得公共系数变量 $x=4.6241$,可得 *book* 出现次数的计算值,$\tilde{C}=Max_{bib-book}\times x=10^5\times x=4.6241\times10^6$,因此,在新书规模小于$3.6241\times10^5$时,均只需局部调整。当新书规模再增大时,进行一次全局调整,即:重新构造方程并求取 $x$ 和 $\tilde{C}$,然后插入新书;以此类推,直到近似解满足 $x-1<0.01$,认为整个图书馆已经饱和。根据仿真实验,图书馆最多可以插入新书$1.9821\times10^7$册,期间共经过8次全局调整,从而 $\theta_i=8/(1.9821\times10^7)=4.036\times10^{-7}$,因此,可以近似认为有

$$Cost_{OIDMaintaining}=[O(\log C)+O(\log\tilde{C})]\times len(E)\qquad(3)$$

对于仿真试验,结合(1)式,有:

$$\begin{aligned}Len(E)&=Len(Editor)\times floor(Max\times x)+Len(title)+Len(publisher)+Len(price)+1\\&=4\times floor(50\times4.6241)+1+1+1+1=928\end{aligned}$$

$\log\tilde{C}=\log(4.6241\times10^6)=6.665$;

因为该例中没有允许出现次数为 $C(C>1)$次的节点,所以 $O(\log C)$这项不存在,从而,根据(3)式有:$Cost_{OIDMaintaining}=6185$。而对于前序算法,在 $D=10^6$的规模下平均插入一本书的调整代价为 $O(D/2)=5\times10^5$。即新算法的代价只有前序算法的1.23%。综合以上分析,新算法有明显优势。

**结论** 更新的代价问题已经成了半结构化文档维护的一个突出问题。本文提出的基于模式的文档顺序维护算法给出了一个有效的解决方案。通过对允许出现多次的元素进行 OID 区间预留,新算法将插入操作时全局范围内的 OID 调整化解为局部的 OID 调整,从而大大降低了文档顺序维护的代价。此外,用户可以基于顺序的标识设计性能优良的索引机制,提高查询操作的效率。在实际应用中,文档模式的特征将直接影响到算法性能,统计信息的利用和缺省倍率的选取都会部分影响到算法性能,我们将在这些方面继续研究。

## 参 考 文 献


1 Li Quanzhong, Moon B. Indexing and Querying XML Data for Regular Path Expressions. In: Proc. of the 27th VLDB Conf. Roma, Italy, 2001

2 XQuery http://www.w3.org/XML/Query

3 Bruno N, Koudas N, Srivastava D. Holistic Twig Joins: Optimal XML Pattern Matching. ACM SIGMOD, June2002

4 Tatarinov I, et al. Updating XML. In: Proc. ACM SIGMOD Int. Conf. on Management of Data, 2001. 413～424

5 图书管理系统. http://www.bn.com/bib.xml

6 McGraw Hill. Data Structures, Algorithms, and Applications in C++. NY, 1998

7 Dietz P F. Maintaining order in a linked list. Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing, San Francisco, California, May 1982. 122～127

8 http://www.w3.org/XML/Schema

9 Standish T A. Data Structure Thechniques. Addison-Wesley Publishing Company, 1980

10 易大义,陈道琦. 数值分析引论. 浙江大学出版社,1998