规则路径表示下 XML 数据查询的动态规划优化方法

洪晓光　李　晖
（山东大学计算机科学与技术学院　济南250100）

摘　要　本文介绍了对有规则路径表示的 XML 数据查询的处理方法，包括传统的处理方法、改进的处理方法及相关的理论基础；然后提出了用动态规划算法对 XML 数据查询的处理进行优化，包括设计步骤和具体算法；最后对全文内容进行了简要总结。

关键词　编码，路径分解，合并连接

The Optimization of Querying XML Data for Regular Path Expressions Based on Dynamic Programming

HONG Xiao-Guang　LI Hui
(Dept. of Computer Science of Shandong University, Jinan 250100)

Abstract　In this paper, several methods are introduced to process querying XML data, which use regular path expressions. Those methods include the conventional approaches and the improved approaches, the theory they based on is also presented . Then, a new algorithm is proposed to optimize the querying XML data for regular path expressions, the proposed algorithm is based on dynamic programming. Finally, the content of this paper is summaried.

Keywords　Numbering scheme, Decomposition of path expressions, Path join

1　引言

随着 XML(Extensible Markup Language)的出现，它已成为 Internet 上信息表示和交换的新标准[1]。因为 XML 数据具有自描述性，它被认为是定义半结构化数据最有前景的方法。为了检索 XML 和半结构化数据，已经出现了几种 XML 查询语言，例如 Lorel[2]，Xml-ql[3]，Xml-gl[4]，Xquery[5]等，它们的共同特征是用规则路径的表示去查询 XML 数据。

以往的研究表明[6]，运用当前关系数据库的技术还不能满足 XML 和半结构化数据的高效存储，并且在处理有规则路径表示的查询时，只采用了一些基于传统的树的遍历的方法，这种方法在查询路径很长或未知的情况下效率很低。Quanzhongli 和 Bongki Moon 在改进的处理方法中[6]，提出了一种新的编码方案，建立了3个主要的索引结构，对查询语句的处理采取了路径分解、合并结果的方法，提高了查询的效率。在合并结果的过程中，特别是对长路径查询的情况，需处理的中间结果多，计算量大，为此本论文提出了用动态规划的算法来确定最优合并次序的方法，并设计出具体算法，来提高查询处理效率。

2　传统的查询处理方法

2.1　Xquery 查询语言的基本符号示意表[5]

符号	功能
-	表示任一个节点
/	表示查询路径中节点的分隔符
|	表示节点的并集
?	表示出现0次或1次的节点
+	表示节点可出现1次以上
*	表示可出现任意次的节点
[]	表示查询的限定条件
@	表示其后面是属性
()	限定优先次序

例如查询语句 Q1：/book/chapter/-*/figure[@caption="apple"]表示是要在所有的 book 下的所有的 chapter 中查找带有属性 caption 值为“apple”的所有 figure。

2.2　处理方法

处理象 Q1这样的有规则路径表示的查询传统的方法[8]是按照 top-down 或 bottom-up 的次序来遍历 XML 文档树。

按 top-down 的次序来处理时，要顺着所有开始于 book 节点的路径，去查找是否有 chapter 的节点来作为后代，这一步要对 XML 数据库中的所有的 book 节点来执行，这就意味着在 XML 树中要遍历从 book 节点到叶节点的每一条可能的路径，如果 book 节点是根节点，那么整棵 XML 树都必须被遍历。

按 bottom-up 的次序来遍历可以降低遍历的成本。对于同样的查询 Q1来讲，所有带有属性 caption 值为“apple”的 figure 节点都要被搜索，从每一个这样的 figure 节点开始向上遍历来确定是否存在 chapter 节点，然后再从每一个这样的 chapter 节点向上遍历来确定是否存在 book 节点作为祖先。这种向上遍历的方法在一般情况下较简单、耗时较小，但对于符合条件的 figure 节点数目很大，而 chapter 节点或 book 节点数目较小的情况，这种遍历方式的成本可能会高于 top-down 方式。

一种折衷的处理方法是同时按照 top-down 和 bottom-up 的次序进行遍历[8]，最终会在路径的某个中间位置相遇，从而得出查询结果。这种方法结合了 top-down 和 bottom-up 的优点，但它的高效性并不总能得到保证，下面改进的方法中采用了路径分解和连接算法来处理，避免了多次遍历树。

3　改进的查询处理方法

3.1　编码方案

XML 的数据对象一般可被模型化为树状结构，其节点代

表元组、属性和文本值，有着父母孩子关系的节点对代表XML数据的嵌套关系。快速地确定在XML树中任一对节点间的祖先后代关系，可加速对规则路径表示查询的处理，树中节点祖先后代关系的确定如通过对节点的编码来判断，可避免大量地对树进行遍历。

3.1.1 Dietz 编码方案[7] 给定树T的任两个节点X和Y，X是Y的一个祖先当且仅当在先序遍历中X出现在Y之前，在后序遍历中X出现在Y之后。如图1所示。

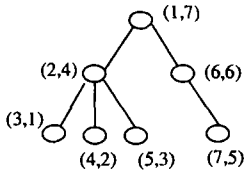

图1

图1中每个节点都被标记为由先序数和后序数组成的数对，如我们可断定节点(1,7)是节点(4,2)的一个祖先，因为在先序遍历中，节点(1,7)出现在节点(4,2)之前(1<4)，在后序遍历中，节点(1,7)出现在节点(4,2)之后(7>2)。这种方法的优点非常明显，即祖先后代关系可通过检查树节点的先序数和后序数的大小在常量时间内被确定。但另一方面此种编码方案不够灵活，因为当树增加一个新节点时，整棵树的编码都要被重新计算。为解决这个问题，Quanzhongli 和 Bongki Moon 利用扩展的先序数和后代范围[6]，提出了下面的编码方案，简称为QB方案。

3.1.2 QB 编码方案 此编码方案将树中的每个节点都用(order,size)数对来标记，如图2所示。它有如下特点：

(1)对树的节点y和它的父母x有，order(x)<order(y)，并且 order(y)+size(y)≤order(x)+size(x)；即区间[order(y),order(y)+size(y)]被包含在区间[order(x),order(x)+size(x)]中。

(2)对树中的任两个兄弟节点x和y，如x是y的兄长，则 order(x)+size(x)<order(y)。

(3)对树的任意节点x，其所有孩子节点设为y，有 size(x)≥Σ,size(y)，size(x)可能为任意大的整数，它要大于节点x当前所有后代节点的size值之和，这样就可以方便地处理未来的插入操作，而不必每次插入都要对树进行重新遍历和编码。

(4)给定树的任两个节点x和y，x是y的一个祖先当且仅当满足条件 order(x)<order(y)≤order(x)+size(x)。

如图2，节点(25,5)包含于(10,30)和(1,100)中，因此order值为25的节点是order值为10和1节点的后代。

"QB编码方案"同"Dietz编码方案"相比，前者能高效地处理动态更新的XML数据，size值为后增加的节点储备了编码空间。改进的查询处理方法就是基于这种编码方案。

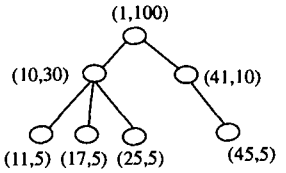

图2

3.2 索引结构

为支持根据元组名、属性名和结构进行查询，即要实现以下三种操作：

(1)给定一元组名字符串，比如说"figure"，去查找所有具有此名的元组，并把它们按所属的文档分类。

(2)给定一属性名字符串，比如说"caption"，去查找所有具有此名的属性，并把它们按所属的文档分类。

(3)给定一个元组，查找它的父元组或孩子元组、属性；给定一个属性，查找它所属的元组。

Quanzhongli 和 Bongki Moon 提出了三个主要的索引[6]：元组索引、属性索引和结构索引。

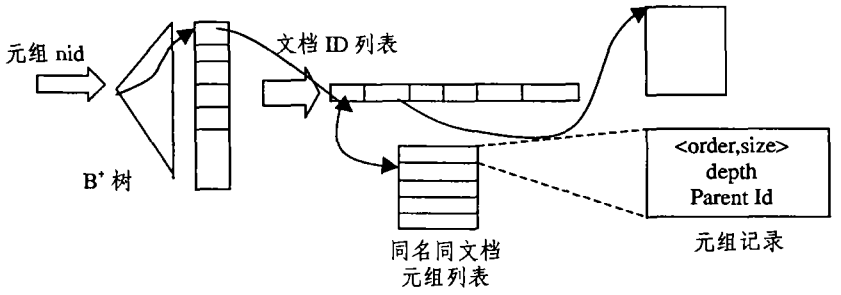

图3 元组索引

(1)元组索引：如图3所示。采用 B^+ 树来实现，利用名字标示(nid)作为键值，其叶节点的每一个入口均指向由固定长度的元组记录组成的集合，这些元组具有相同的名字串，并按它们所属的文档进行分组。利用元组索引可使我们快速地查找出拥有相同名字的所有元组，这是在处理有规则路径表示的查询中最普遍的操作。每一个元组记录都包含数对(order,size)和其他相关信息，记录是按照order值来排序的。

(2)属性索引：结构同元组索引类似，不同之处是在记录中包含一个值标示(vid)，它被当作一个键用于从值表中获得属性值。

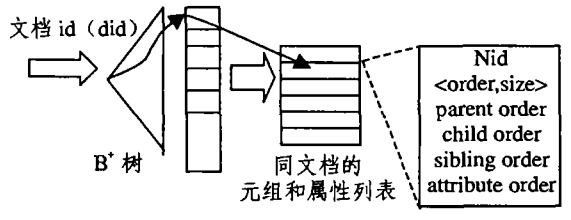

图4 结构索引

(3)结构索引：如图4所示。它由线性矩阵组成，每一个线

性矩阵都用来存储元组或属性的固定长度记录，这些元组或属性来自于同一个 XML 文档。在一个线性矩阵中，元组和属性按照它们的 order 值被一起存储。在每一个记录中，存放有一个名字标示(nid)、第一个兄弟的 order 值、第一个孩子节点、第一个属性节点等。

3.3 查询表达式的分解与连接算法[6]

Quanzhongli 和 Bongki Moon 将规则路径查询表达式分解为以下五种基本表达式：

(1)只由一个元素或一个属性组成的表达式；

(2)由一个元素和一个属性组成的表达式(例如 figure[@caption="abc"])；

(3)由两个元素组成的表达式(例如 chapter/-*/figure 或 chapter/figure)；

(4)一个子表达式的 kleene 闭包；

(5)两个子表达式的并集。

对第(1)种情况可利用元组或属性的索引直接得到结果，第(5)种情况可通过组合两个中间结果或直接按文件分组来处理，对第(2)、(3)、(4)种情况需分别采用如下三个算法来进行中间结果的连接合并：

(1)∈A-Join 算法(用于元素集合和属性集合的合并，对应于由一个元素和一个属性组成的子表达式)：

```
输入:{E1,…,Em},Ei 表示拥有相同文件标示符(did)的元素集
     {A1,…,An},Aj 表示拥有相同文件标示符(did)的属性集
输出:元素为(e,a)的集合,要求满足 a 是 e 的属性
//据文件标示符来分类合并集合{Ei}和{Aj}
1:for each Ei and Aj with the same did do
  //据孩子父母关系分类合并 Ei 和 Aj
2:      for each e∈Ei and a∈Aj do
3:          if (e is a parent of a )then output(e,a)
      end
end
```

(2)∈∈-Join 算法(用于两个元素集合的合并，对应于由两个元素组成的表达式)：

```
输入:{E1,…,Em}和{F1,…,Fn},Ei 和 Fj 是拥有相同文件标示符
     (did)的元素集
输出:(e,f)的集合,要求满足 e 是 f 的祖先
//据文件标示符来分类合并集合{Ei}和{Fj}
1:for each Ei and Aj with the same did do
  //据祖先后代关系分类合并 Ei 和 Aj
2:    for each e∈Ei and f∈Fj do
3:        if (e is an ancestor of f )then output(e,f)
    end
end
```

(3)kleene closure 算法(对应于求一个子表达式的 kleene 闭包)：

```
输入:{E1,…,Em},Ei 是一个 XML 文档的一组元素
输出:{E1,…,Em}的 Kleene 闭包
//重复执行∈∈-Join 算法
1:set i=1
2:set K1=(E1,…,Em);
3:repeat
4:    set i=i+1
5:    set Ki= ∈∈-Join(Ki-1,K1)
 until (Ki is empty);
6:  output Union of K1,K2,…,Ki-1;
```

4 利用动态规划算法进行 XML 数据的查询优化

在利用上述算法对中间结果集合进行连接合并时，由于各集合中元素个数不同，连接合并的次序在很大程度上决定着计算量的大小，决定着查询的执行效率，本节将讨论用动态规划的算法来确定出一个最优的合并连接次序，来实现对有规则路径表示的 XML 数据查询的优化。

4.1 动态规划算法的基本要素及求解步骤

动态规划算法与分治法类似，其基本思想也是将待求解的问题分解成若干子问题，先求解子问题，然后从这些子问题的解得到原问题的解。但不同的是适合用动态规划法求解的问题，经分解得到的子问题往往不是相互独立的。若用分治法来解这类问题，则分解得到的子问题太多，以至于解决原问题需要耗费指数时间。

动态规划算法的基本思路是用一个表来记录所有已解决的子问题的答案，不管该子问题以后是否被用到，只要它被计算过，就将其结果填入表中，在需要时再找出，这样就可避免大量的重复计算。

动态规划算法通常用于求解具有某种最优性质的问题。这类问题中，可能会有许多可行解，每个解都对应一个值，我们希望找到最优值(最大值或最优值)的那个解。设计一个动态规划算法，通常可按以下几个步骤进行：

(1)找出最优解的性质，并刻画其结构特征。

(2)递归地定义最优值。

(3)以自底向上的方式计算出最优值。

(4)根据计算最优值时得到的信息，构造一个最优解。

动态规划算法的有效性依赖于问题本身所具有的两个重要性质：最优子结构和子问题重叠性质，问题所具有的这两个重要性质是该问题可利用动态规划算法求解的基本要素。最优子结构性质使我们能够以自底向上的方式递归地从子问题的最优解逐步构造出整个问题的最优解。同时，它也使我们在相对小的子问题空间中考虑问题。在用递归算法自顶向下解此问题时，每次产生的子问题并不总是新问题，有些子问题被反复计算多次，动态规划法正是利用了这种子问题的重叠性质，对一个子问题只解一次，而后将其保存在一个表格中，当再次需要解此问题时，只是简单地用常数时间查看一下结果。通常，不同的子问题个数随输入问题的大小呈多项式增长。因此，用动态规划算法通常只需要多项式时间，从而获得较高的解题效率。

4.2 用动态规划算法进行中间结果的连接合并

在用3.3节中的三个连接算法进行中间结果的合并时，不同的合并次序会导致不同的计算量。例如：设一 XML 数据查询表达式经分解后产生3个中间结果集合，分别为 A(100)、B(10)、C(5)，括号中的数字为其含有的元素个数，设一般情况下，两集合通过上述算法合并后得到的集合元素个数最大为两者元素个数的最大值。这三个集合有两种合并次序为：(1)(AB)C (2)A(BC)。第一种计算量为：100×10+5×100=1500，第二种计算量为：10×5+10×100=1050，可见第一种次序的计算量是第二种的1.43倍。

4.2.1 *利用穷举算法确定合并的最优次序*

如何确定合并的最优次序，使得合并的计算量最小，穷举搜索法是最容易想到的方法，算法列出所有可能的合并次序，并计算出每种计算次序相应的计算量，由此找出一种所需计算次数最小的合并次序。然而，这种方法计算量太大，如对于 n 个中间结果集合的合并，设有 P(n)个不同的计算次序，由于可以先在第 k 个和第 k+1个集合之间将原集合序列分为两个集合子序列(k=1,2,…,n-1)，然后分别对这两个集合子序列进行最优次序确定，以此类推，最终得到原集合序列的最优合并次序，由此可得关于 P(n)的递归式如下：

$$P(n)=\begin{cases}1 & n=1\\ \sum_{k=1}^{n-1}P(k)P(n-k) & n>1\end{cases}$$

解此递归方程可得，P(n)实际上是 Catalan 数，即 P(n)=C(n-1)，其中 $C(n)=\frac{1}{n+1}\binom{2n}{n}=\Omega(4^n/n^{3/2})$，即 P(n)

是随 n 的增长呈指数增长的，所以穷举搜索法不是一个有效算法。

4.2.2 利用动态规划算法确定最优合并次序

(1)分析最优解的结构。先来分析要求解问题的最优解结构特征，为方便起见，将需合并的中间结果集合 $T_iT_{i+1}\cdots T_j$ 简记为 T[i:j]。下面来计算合并集合 T[1:n]的一个最优次序。设这个计算次序在矩阵 A_k 和 A_{k+1} 之间将集合链断开，$1\leqslant k<n$，则加括号方式为$(T_1\cdots T_k)(T_{k+1}\cdots T_n)$，以此类推，先分别计算 T[1:k]和 T[k+1:n]，然后将计算结果再合并得到 T[1:n]，总计算量为 T[1:k]的计算量加上 T[k+1:n]的计算量，再加上 T[1:k]和 T[k+1:n]相合并的计算量。

此问题的一个关键特征是：合并集合链 T[1:n]的一个最优次序所包含的合并集合子链 T[1:k]和 T[k+1:n]的次序也是最优的。可用反证法来证明：假设有一个合并 T[1:k]的次序需要的计算量更少，则用此次序替换原来合并 T[1:k]的次序得到的合并 T[1:n]的次序需要的计算量将比最优次序所需计算量更少，由此得出矛盾。同理可知，合并集合链 T[1:n]的一个最优次序所包含的合并集合子链 T[k+1:n]的次序也是最优的。因此，处理规则路径表示的 XML 数据的查询时，中间结果集合的连接合并次序问题的最优解包含着其子问题的最优解，这种性质称为最优子结构性质，是可用动态规划算法求解的显著特征。

(2)建立递归关系。设合并集合 T[i:j]，$1\leqslant i\leqslant j\leqslant n$ 所需的最小计算次数为 m[i][j]，一般情况下，合并产生的新集合中元素个数为 $N[i][j]=MAX\{N_k\}$，$i\leqslant k\leqslant j$，N_k 表示集合 T_k 中元素个数。设合并集合 T[1:n]的最优值为 m[1][n]

当 i=j 时，$T[i:j]=T_i$ 为单个集合，无需计算，因此 m[i][i]=0，i=1,2,…,n。

当 i<j 时，可利用最优子结构性质来计算 m[i][j]。若合并 T[i:j]的最优次序在 T_k 和 T_{k+1} 之间断开，$i\leqslant k<j$，则 m[i][j]=m[i][k]+m[k+1][j]+N[i][k]*N[k+1][j]，由于计算时并不知道断开点 k 的位置，所以 k 还未定，不过 k 的位置只有 j−i 种可能，即 $k\in\{i,i+1,\cdots,j-1\}$，所以 k 是这 j−i 个位置中使计算量达到最小的那个位置，从而 m[i][j]可递归地定义为：

$$m[i][j]=\begin{cases}0 & i=j\\ \min\limits_{i\leqslant k<j}\{m[i][k]+m[k+1][j]+N[i][k]*N[k+1][j]\} & \end{cases}$$

若将对应于 m[i][j]的断开位置 k 记为 S[i][j]，在计算出最优值 m[i][j]后，可递归地由 S[i][j]构造出相应的最优解。

(3)计算最优值。据计算 m[i][j]的递归式，可写一个递归算法来计算 m[1][n]。简单的递归算法将耗费指数计算时间，因为对于$1\leqslant i\leqslant j\leqslant n$ 不同的有序对(i,j)对应于不同的子问题，所以不同子问题的个数最多只有$\binom{n}{2}+n=\theta(n^2)$个。所以在递归计算时，许多子问题被重复计算多次，用动态规划法处理时，可根据递归式以自底向上的方式进行计算，在计算过程中，保存已解决的子问题答案，每个子问题只计算一次，需要时只需查一下，从而避免了大量的重复计算，最终得到多项式时间的算法。下面给出的合并集合链 m[1][n]的动态规划算法 Dynamicjoin，算法除了输出最优值数组 m 外，还输出记录最优断开位置的数组 s。

输入各集合的元素个数到数组 P[0]到 P[n−1]中。

数组 N[i][j]用来记录合并集合链 $T_i\cdots T_j$ 后所得新集合的元素个数，一般情况下认为新集合元素个数为 T_i 到 T_j 中元素个数的最大者。

数组 m[i][j]用来记录合并集合链 $T_i\cdots T_j$ 所需的最小计算次数。

数组 s[i][j]用来记录合并集合链 $T_i\cdots T_j$ 时的最优断开位置。

算法如下：

```
Dynamicjoin (int p,int n,int * *m,int * *s)
{
 /*只有一个集合作连接时，计算次数为零*/
 for (int i=1;i<=n;i++)m[i][i]=0;
 /*计算出数组 N[i][j]的值备用*/
   for (int i=1;i<=n;i++)
     { for (int j=i;j<=n;j++)
       { if (i==j){N[i][j]=p[i-1]}
         else{N[i][j]=p[i-1];
              for (int js=i+1;js<=j;js++)
                {if (p[js-1]>N[i][j])N[i][j]=p[js-1]}
            }
       }
 }
/*集合链长度从2到 n，分别确定计算量最小的断开位置 k，存放在数组 s[i][j]中*/
  for(int r=2;r<=n;r++)
    for(int i=1;r<=n-r+1;i++)
    {
      int j=i+r-1;
      m[i][j]=m[i+1][j]+N[i][i]*N[i+1][j];
      s[i][j]=i;/*k=i 时*/
       for(int k=i+1;k<j;k++)/*对 k 的取值从 i+1到 j 中取值，确定最优断点*/
       { int t=m[i][k]+m[k+1][j]+N[i][k]*N[k+1][j];
         if(t<m[i][j]){m[i][j]=t;s[i][j]=k;}
       }
    }
}/*end of Dynamicjoin*/
```

算法 Dynamicjoin 首先计算出数组 N[i][j]以备用，其中 i=1,2,…,n;j=1,2,…,n。然后计算出 m[i][i]=0,i=1,2,…,n，再根据递归式，按集合链长递增的方式依次计算 m[i][i+1],i=1,2,…,n−1(集合链长度为2);m[i][i+2],i=1,2,…,n−2(集合链长度为3);…。在计算 m[i][j]时，只用到已计算出的 m[i][k]和 m[k+1][j]。

例 设要合并6个中间结果集合 $T_1T_2T_3T_4T_5T_6$，各集合的元素个数分别为 P[0]=2000 P[1]=80 P[2]=1000 P[3]=90 P[4]=800 P[5]=3000。经算法 Dynamicjoin 处理后，得出的数组 N[i][j]、m[i][j]和 s[i][j]如表1、2、3所示。

表1 N[i][j]

	j:	1	2	3	4	5	6
i:	1	2000	2000	2000	2000	2000	3000
	2		80	1000	1000	1000	3000
	3			1000	1000	1000	3000
	4				90	800	3000
	5					800	3000
	6						3000

表2 s[i][j]

	j:	1	2	3	4	5	6
i:	1	0	1	1	1	1	5
	2		0	2	2	2	5
	3			0	3	3	5
	4				0	4	5
	5					0	5
	6						0

表3 m[i][j]

	j:	1	2	3	4	5	6
i:	1	0	160000	208000	217000	295200	8952000
	2		0	80000	170000	952000	3952000
	3			0	90000	872000	3872000
	4				0	72000	2472000
	5					0	2400000
	6						0

例如在计算 m[1][5]时，依递归式有

$$m[1][5]=\min\begin{cases} m[1][1]+m[2][5]+N[1][1]*N[2][5]=2952000 \\ m[1][2]+m[3][5]+N[1][2]*N[3][5]=3032000 \\ m[1][3]+m[4][5]+N[1][3]*N[4][5]=3752000 \\ m[1][4]+m[5][5]+N[1][4]*N[5][5]=3770000 \end{cases}=2952000$$

且 k=1，因此 s[1][5]=1。

(4)构造最优解。算法 Dynamicjoin 只是计算出了最优值，并未给出最优解，也就是说，我们只得出要合并所给的集合链所需的最小计算量，还需进一步确定按什么次序来合并才能达到此最小计算量。算法 Dynamicjoin 已记录了构造一个最优解所需的全部信息，s[i][j]中的值 k 表示合并集合链 T[i:j]的最佳方式应在集合 Tk 和 Tk+1之间断开，即合并次序为(T[i:k])(T[k+1:j])。因此，从数组 s 中记录的值可知合并 T[1:n]的最优方式为(T[1:s[1][n]](T[s[1][n]+1:n])。而 T[1:s[1][n]]的最优合并方式为(T[1:s[1][s[1][n]]])(T[s[1][s[1][n]]+1:s[1][n]]；同理可确定 T[s[1][n]+1:n]的最优合并方式为在 s[s[1][n]+1][n]处断开……照此递推，最终可得集合链 T[1:n]的最优合并次序，即构造出问题的一个最优解。

下面的算法 Traceback 按算法 Dynamicjoin 计算出的断点矩阵 s 的值输出合并集合链 T[1:n]的最优次序。

```
void traceback(int i,int j,int * *s)
{
  if(i==j)return;
  Traceback(i,s[i][j],s);
  Traceback(s[i][j]+1,j,s);
  cout<<"Multiply T"<<i<<","<(s[i][j];
  cout<<"and T"<<(s[i][j]+1)<<",")
  <<endl
}
```

要输出 T[1:n]的最优合并次序只要调用 Traceback(1,n,s)即可。对于上例，通过调用 Traceback(1,6,s)，得到最优合并次序为：$(T_1(T_2(T_3(T_4T_5))))T_6$。

(5)算法 Dynamicjoin 复杂性分析。本算法的主要计算量取决于程序中对 r,i 和 k 的三重循环，循环体内的计算量为 O(1)，而三重循环的总次数为 $O(n^3)$，因此该算法的计算时间上界为 $O(n^3)$，算法占用的空间为 $O(n^2)$。由此可见，动态规划算法比穷举搜索法要有效得多。

结束语 随着 XML 作为 Internet 上数据表示和交换标准的出现，如何高效地进行 XML 数据的查询已变得越来越重要，本文介绍的分解合并方法，在处理有规则路径表示的 XML 数据的查询时，与传统的处理方法相比，效率有很大提高；在将动态规划算法用于中间结果集合的合并处理后，又在很大程度上加快了查询的处理速度，提高了查询效率。

参 考 文 献

1 Bray T, et al. Extensible markup language (XML) 1.0 second edition W3C recommendation: [Technical Report REC-xml-20001006]. World Wide Web Consortium, Oct. 2000
2 Abiteboul S, et al. The Lorel query language for semistructured data. International Journal on DigitalLibraries, 1997, 1(1): 68～88
3 Deutsch A, et al. A query language for XML. In: Proc. of the 8th Intl. World Wide Web Conf. Toronto, Canada, May 1999. 77～91
4 Ceri S, et al. XML-GL: A graphical language for querying and restructuring XML documents. In: Proc. of the 8th Intl. World Wide Web Conf. Toronto, Canada, may 1999. 93～109
5 Chamberlin D, et al. Xquery: A Query Language for XML W3C working draft: [Technical Report WD-xquery-20010215]. World Wide Web Consortium, Feb. 2001
6 Li Quanzhong, Moon B. Indexing and Querying XML Data for Regular Path Expressions. In: Proc. of the 27th VLDB Conf. Roma, Italy, 2001
7 Dietz P F. Maintaining order in a linked list. In: Proc. of the Fourteenth Annual ACM Symposium on Theory of Computing, San Francisco, California, May 1982. 122～127
8 McHugh J, Widom J. Query optimization for XML. In: Proc. of the 25th VLDB Conf. Edinburgh, Scotland, Sep. 1999. 315～326

(上接第79页)

```
edge_table_name (edge_info_column.geometry) INDEXTYPE IS
    MDSYS.SPATIAL_INDEX;
CREATE INDEX rtindex_node ON
node_table_name (node_info_column.position) INDEXTYPE IS
    MDSYS.SPATIAL_INDEX;
```

而对于 FACE，实际上并不存在一个 SDO_GEOMETRY 类型的字段。如果要对它建立索引，需要用到 Oracle Spatial 9i 的一个新功能，即建立基于函数的索引[1]，针对前面定义的 get_face()函数返回的 SDO_GEOMETRY 类型来建立空间索引：

```
CREATE INDEX rtindex_face ON face_table_name(get_face
(face_table_name, edge_id_column)) INDEXTYPE IS
MDSYS.SPATIAL_INDEX;
```

需要说明的是，如果要建立空间索引，Oracle Spatial 要求用于建索引的 sdo_geometry 字段在 user_sdo_geom_metadata 表中注册。所以，必须在该表中插入新的记录：

```
INSERT INTO user_sdo_geom_metadata VALUES (
edge_table_name, '
edge_info_column.geometry',
MDSYS.SDO_DIM_ARRAY (MDSYS.SDO_DIM_ELEMENT
('x', 110, 120, 0.1), MDSYS.SDO_DIM_ELEMENT('y', 30,
40, 0.1)), 8307);
```

结语 本文提出了在 Oracle 下进行拓扑管理的框架，它实现了在大型的 RDBMS 中存贮和管理空间拓扑的基本的模型。这种方案的基本思路就是使用 Oracle 提供的面向对象能力来定义拓扑对象，使用元数据表来建立拓扑相关表之间的关联，使用过程语言 PL/SQL 在服务器端实现基本的空间查询分析功能，同时，在必要的时候建立视图来提供更好的概念表达。

基于该模型，只需使用简单的 SQL 语句就可以进行常规的一些空间查询、分析，而对于该模型不同直接提供的空间处理能力，比如最短路径分析，可以通过编写 PL/SQL 过程或者使用 Oracle 的服务器端的 Java 扩展功能来实现，这个过程实际上就是将算法移植到 Oracle 服务器中的过程。

作为一个基本的框架，本文中没有提及在数据更新的情况下，如何维护拓扑关系一致性的问题。这也是我们以后需要研究的重点内容。从技术上讲，通过事务处理、触发器机制以及 Oracle 对 Java 语言的强大支持能力，可以在服务器端实现数据更新和拓扑关系维护。我们将继续在这方面进行更进一步的探索。

本文中提出的实现模型也可以扩展到其他的一些 RDBMS 平台上，比如 Informix、DB2等。

参 考 文 献

1 Oracle Corporation. Oracle9i SQL Reference Release 2 (9.2), March2002
2 Oracle Corporation. Oracle Spaital User's Guide and Reference Release 9.2, March2002
3 Watson P. Topology and ORDBMS technology. White Paper of Laser Scan Ltd, 2002
4 黄杏元，马劲松，汤勤. 地理信息系统概论(修订版). 北京：高等教育出版社，2001