FDewey Encoding: an Approach Based on Dewey for Storing an XML Document into Database

Irvanizam
Department of Informatics, Syiah Kuala University, Darussalam-Banda Aceh
irvanizam.zamanhuri@informatika.unsyiah.ac.id

Abstract. A technique of labeling node in a tree is necessarily implemented to perform queries in an XML data structure. This technique can be used because an XML document is able to be transformed into an XML tree. Dewey is one of old techniques for labeling node that re-labels some sibling nodes when a new node is inserted into the XML tree. In this paper, we propose an extended dewey approach, namely FDewey Encoding, to minimalize relabeling node when a new node in the tree is added. For a new node located in between two consecutive nodes, this encoding calculates the mean of those two nodes, and then uses it as a new label postfix. Relabeling node simply occurs when the mean value is smaller than a given epsilon value.

Keywords: Dewey, FDewey, Insert, XML.

INTRODUCTION

Extensible Markup Language (XML) is a markup language that can be used as a standard document to exchange information through the internet [1,2,3,9]. For instance, providing a web service using XML document and messaging system has encoded all communications between a client and server. Besides protecting the servers, a client could also share data over computer networks without changing data in one server with the other. An XML document has a data structure likely hierarchy, and then can be viewed as an ordered tree [1,2,4]. As an order tree, the document must be able to recognize some tree properties or node relationships such as Parent/Child (PC), Ancestor/Descendant (AD), siblings (S), and node order through its elements. Therefore, a technique to label elements is necessarily defined to handle this limitation.

According to papers [2,9], Dewey encoding directly adopts query process in an XML document. Each document node represents a path from the node to the tree root. In spite of the fact that the nodes relationships can be easily identified, the encoding still requires high cost to insert and delete a node into the tree. If a node is inserted or deleted, the encoding will replace some labels of all its right sibling nodes. Now days, some labeling techniques [4,7,8] have been introduced by researchers for processing XML queries. One of ways to obtain it is that each unique label of the XML document is stored into a relational database. Then, by using Structure Query Language (SQL), the result of the queries would dynamically be obtained from the database.

In this paper, a Dewey concept will be re-demonstrated how to insert and delete a node into a tree. As comparison, we also propose an extended Dewey approach, namely FDewey Encoding. For general overview, the primary discussions for this research are repeated here:

- To show a technique for labeling node using FDewey, and to explain how nodes are periodically relabeled.
- To demonstrate some advantages of both Dewey and FDewey Encoding.
- To display insert and delete performances for Dewey and FDewey using graphics.

The rest of the paper is organized as follows: Section 2 presents a Dewey Encoding concept. Section 3 introduces an approach of FDewey Encoding. Section 4 describes results of the experiment, and in the end of the paper, Section 5 concludes the paper.

DEWEY ENCODING

[1] has introduced a Dewey Encoding that can identify XML element positions. Every XML element is an element node labeled as a following string: if a node is root, Dewey will label it with string ‘1’, otherwise, Dewey writes it with \( \text{label}(u) = \text{label}(s).x \), where \( u \) is child of the element \( s \), and \( x \) is a child position of the root. By using this label, it
is easy to determine a relationship between one node to another. For instance, the element $u$ is an ancestor of the elements $s$ if only if $label(u)$ is a prefix from $label(s)$.

Figure 1.a and 2.b show a simple XML document, and a tree that has been transformed using Dewey Encoding, respectively. A dewey label has some tree properties. Let $T$ be a Dewey tree with label of node $A$: $a_1, a_2, a_3, ..., a_m$, and label of node $B$: $b_1, b_2, b_3, ..., b_n$. The tree properties (P) or node relationships such as ancestor, parent, and sibling are defined as follows:

- **P1 (ancestor)**: $A$ is ancestor of $B$ if only if $m < n$, and $a_1 = b_1, a_2 = b_2, ..., a_m = b_m$.
- **P2 (parent)**: $A$ is parent of $B$ if only if $A$ is ancestor of $B$, and $m = n - 1$, and $a_1 = b_1, a_2 = b_2, ..., a_m = b_m$.
- **P3 (sibling)**: $A$ is sibling of $B$ if only if parent of $A$ is the same with parent of $B$.

Assuming that there is a dewey label of element $u$ with ‘1.2.3.4’, so the parent of the element $u$ is ‘1.2.3’, the grandparent of the element $u$ is ‘1.2’, and one of siblings for the element $u$ is ‘1.2.3.5’. Figure 1.b describes that the elements with label ‘1.1.2’, ‘1.1’, and ‘1’ are the ancestor of element ‘1.1.2.1’. The parent of element ‘1.1.2.1’ is the element ‘1.1.2’, whereas the sibling of the element ‘1.1.2.1’ is the element ‘1.1.2.2’.

Dewey is not only able to identify some node relationships, but it can also sort all XML nodes. According to Xu et al. [6] all nodes that have been labeled by Dewey in a tree can be ordered using Dewey order. All nodes are traversed from root using pre-order. Supposing that there are two dewey labels $A$: $a_1, a_2, ..., a_m$ and $B$: $b_1, b_2, b_3, ..., b_n$, so Dewey order is scientifically defined as follows:

**DEFINITION 1 (DEWEY ORDER)**

$A <_{\text{dewey}} B$ if only if one of the following condition is satisfied:

- **Condition 1.** $m < n$ and $a_1 = b_1, a_2 = b_2, ..., a_m = b_m$.
- **Condition 2.** $\exists k \leq \min(m,n)$, such that $a_1 = b_1, a_2 = b_2, ..., a_{k-1} = b_{k-1}$ and $a_k < b_k$.

By using Definition 1 and Figure 1.b, we can see that after reading the label ‘1.1.1’, the XML document will read label ‘1.1.2.1’ (“1.1.1” $<_{\text{dewey}}”1.1.2.1””). Another example, the element with label ‘1.2.1’ is an element that appears before the element ‘1.2.2’ (“1.2.1” $<_{\text{dewey}}”1.2.2”$).
Inserting and Deleting Nodes using Dewey Encoding

Inserting or deleting a node is a query process that frequently occurs in XML tree. The processes are performed by adding or deleting a part of elements in XML tree, Dewey then reconstructs the tree by replacing label of nodes such that the tree can be viewed again as a new Dewey tree. In general, there are three locations where Dewey is able to insert or delete a node. Figure 2.a and 2.b demonstrate those two processes.

Figure 2.a illustrates a new node with label '1.1.2.1' (baru) added into the tree (in Figure 1.b), and then becomes the first child of the node ‘1.1.2’. Therefore, the label of the node ‘1.1.2.1’ (first) and ‘1.1.2.2’ (second) are replaced by ‘1.1.2.2’ and ‘1.1.2.3’ respectively. If a new node is labeled by ‘1.2.1’ (as the first child of the node ‘1.1’), then the five next nodes ‘1.2.1’, ‘1.2.1.1’, ‘1.2.1.2’, ‘1.2.2’, and ‘1.2.3’ are sequentially changed.

Another case, inserting a node at the right most children of a parent is performed by appending an integer number to the parent label, which is the last child of the parent. For instance, the node ‘1.1.5’ is efficiently inserted at the right position of the node ‘1.1.4’ without replacing some other nodes, and is the last child of the parent node ‘1.1’.

Deleting a node in the Dewey tree is the inverse of inserting nodes [6,9]. When the inserted node ‘1.1.2.1’ at the tree shown Figure 2.a is removed, then the tree will be back like Figure 1.b. Similarly, if the node ‘1.1.2.1’ shown at Figure 2.a is deleted, then the node ‘1.1.2.2’ and ‘1.1.2.3’ will be respectively replaced back to ‘1.1.2.1’ and ‘1.1.2.2’. In addition, when deleting the node ‘1.1’ from the tree is occurring, almost all nodes in the tree (excluding the root) will be re-written their labels. Figure 2.b demonstrates how to remove the node ‘1.1.2.1’ and ‘1.1’ at the tree (Figure 2.a).

FDEWEY ENCODING

Float Dewey (FDewey) Encoding is a proposed technique based on Dewey approach for labeling nodes using decimal numbers and characters ‘*’-s to separate among node label components. This technique is similar to Dewey at an initial level. After inserting a new node, FDewey appends the prefix of the inserted node with a decimal number. Inserting a new node can be implemented at the leftmost, the rightmost, and between two consecutive nodes.

DEFINITION 2 (INSERT FDEWEY)

Let $T$ be a FDEWEY Tree with label of node $A$: $a_1 * a_2 * a_3 * \ldots * a_m$, $B$: $b_1 * b_2 * b_3 * \ldots * b_m$, and $P$: $a_1 * a_2 * a_3 * \ldots * a_{m-1}$ where $a_1 = b_1, a_2 = b_2, \ldots, a_{m-1} = b_{m-1}$, $a_m < b_m$. $A$ is the first child of parent $P$, $B$ is the last child of parent $P$, then label of node $C$ can be added by satisfying one of the following conditions:
An XML tree consists of at least two nodes and a single root. Each node or root may have either empty or a set of children nodes in which a new node could be inserted between two consecutive children nodes. A node has a certain distance with its nearest sibling. In FDeWey, a distance between two nodes is an important issue because the process of re-labeling some nodes depends on the distance among nodes and a given epsilon value.

**DEFINITION 3 (NODE DISTANCE)**

Let $N_{k-1} = a_1^*a_2^*a_3^*...*a_m$, and $N_k = b_1^*b_2^*b_3^*...*b_m$, are two consecutive nodes and two children of node $A$: $a_1^*a_2^*a_3^*...*a_m$, then the distance between node $N_{k-1}$ and $N_k$ is defined as follows:

$$
\text{Distance} (N_{k-1}, N_k) = \text{MIN} (|a_m - b_m|, b_m)
$$

Inserting nodes at the rightmost and deleting nodes are similar to Dewey where the FDeWey does not need to calculate the node distance. However, once a new node will be inserted at the leftmost or between two consecutive nodes, FDeWey calculates the node distance in order to validate whether the distance is smaller than the given epsilon value or not. If so, the encoding will re-label the inserted node’s siblings and their descendants. Figure 3.a, 3.b, and 3.c illustrate how to insert a node at the three locations using FDeWey.

![Figure 3](image-url)

**FIGURE 3.** Inserting and Removing a Node using FDeWey Encoding

Re-labeling some nodes is infrequently performed in a FDeWey tree. With giving a very small epsilon value ($\epsilon$), there will exist many possibilities for placing many nodes at between two consecutive nodes. Supposing that the
epsilon is equal to 0.1, so there has possibly ten nodes that can be inserted at between the node ‘1*1’ and ‘1.2’. The number of places for new nodes depends on how small the given epsilon value is. Once a new node is added as the nearest right sibling of the node ‘1*1’, many nodes which are the siblings of the inserted node and their descendants will be replaced by other FDewey labels. Figure 3.d show to modify some node labels for inserting a node.

RESULTS

The results of an experimental study designed to compare the insert and delete performances of the two encoding introduced in this paper will be described in this section. For the experiments, the three freely XML datasets dblp, nasa, and [5] shown in Table 1 were stored into a relational databases using Dewey, and another using FDewey. Each database has eight tables with three attributes, shown in Table 2. One thousand tuples inserted randomly into each table were separately calculated their average times to compare the insert performance. Similarly, one thousand arbitrary tuples existing in each table of the two databases were removed and then the average times obtained from this process were tabulated into a separate graphic. All experiments were run on a personal computer with 1333 MHz Intel Core i5 processor with 4GB of physical memory running under Mac OSX version 10.9.4.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Size (MB)</th>
<th>Number of Node</th>
<th>Max/Avg depth</th>
<th>Max/Avg funout</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBLP-1</td>
<td>0.1</td>
<td>2K</td>
<td>2/1</td>
<td>199/1999</td>
</tr>
<tr>
<td>DBLP-2</td>
<td>1.5</td>
<td>30K</td>
<td>3/3</td>
<td>974/56</td>
</tr>
<tr>
<td>DBLP-3</td>
<td>4.1</td>
<td>79K</td>
<td>3/3</td>
<td>8662/107</td>
</tr>
<tr>
<td>DBLP-4</td>
<td>7.2</td>
<td>139K</td>
<td>3/3</td>
<td>15K/107</td>
</tr>
<tr>
<td>Nasa</td>
<td>23.8</td>
<td>533K</td>
<td>10/7</td>
<td>2K/225</td>
</tr>
<tr>
<td>Treebank</td>
<td>85.4</td>
<td>2M</td>
<td>36/8</td>
<td>56K/1.6K</td>
</tr>
<tr>
<td>DBLP-5</td>
<td>260</td>
<td>5M</td>
<td>3/3</td>
<td>79K/2K</td>
</tr>
<tr>
<td>DBLP-6</td>
<td>592</td>
<td>11M</td>
<td>3/3</td>
<td>92K/3K</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Node ID</th>
<th>Label</th>
<th>Element Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>dblp</td>
</tr>
<tr>
<td>2</td>
<td>1*1</td>
<td>book</td>
</tr>
<tr>
<td>3</td>
<td>1<em>1</em>1</td>
<td>author</td>
</tr>
<tr>
<td>4</td>
<td>1<em>1</em>2</td>
<td>author</td>
</tr>
<tr>
<td>5</td>
<td>1<em>1</em>2*1</td>
<td>first</td>
</tr>
<tr>
<td>6</td>
<td>1<em>1</em>2*2</td>
<td>second</td>
</tr>
<tr>
<td>7</td>
<td>1<em>1</em>3</td>
<td>title</td>
</tr>
</tbody>
</table>

The results of comparing the insert and delete performances can be shown in Figure 4.a and 4.b. The insert performance of Dewey depends on the number of elements, depth, and fannout of a tree. The average time of insertion will be bigger if the number of elements, depth, and fannout used by the datasets are also bigger. Meanwhile, FDewey does not show that there is dependency on either the number of elements or the number of depth, or fannout. Interestingly, there is a slight difference in the delete performance. FDewey is a little bit faster than Dewey on the shallow datasets. This is because those datasets have no descendants and FDewey does not need to replace the label of the deleted node siblings.
CONCLUSION

A node labeling technique is an important feature of the XML data model, it might or might not provide a good insert or delete performance using a relational database system to store large XML documents. This will be a challenge to discover any other techniques that can demonstrate an efficient way to insert or delete elements. Based on the results of this experimental study, this paper shows that FDewey performs best on insert, while Dewey still performs best on delete at a position where a deleted node has no children.

ACKNOWLEDGMENTS

Working on this research with large datasets has been a wonderful experience. There are two persons who have supported me to accomplish this paper. In this section, I am grateful to Michael Boehlen who has introduced this research field and Nicolas Augsten who has taught me courses related to this research topic.

REFERENCES