L-Tree: A Dynamic Labeling Structure for Ordered XML Data

Yi Chen¹, George Mihaila², Rajesh Bordawekar², and Sriram Padmanabhan²

¹ University of Pennsylvania
yicn@seas.upenn.edu
² IBM T.J. Watson Research Center
{mihaila,srp,bordaw}@us.ibm.com

Abstract. With the ever growing use of XML as a data representation format, we see an increasing need for robust, high performance XML database systems. While most of the recent work focuses on efficient XML query processing, XML databases also need to support efficient updates. To speed up query processing, various labeling schemes have been proposed. However, the vast majority of these schemes have poor update performance. In this paper, we introduce a dynamic labeling structure for XML data: L-Tree and its order-preserving labeling scheme with O(log n) amortized update cost and O(log n) bits per label. L-Tree has good performance on updates without compromising the performance of query processing. We present the update algorithm for L-Tree and analyze its complexity.

1 Introduction

With the advent of XML as a data representation format, we see an increasing need for robust, high performance XML database management systems which support efficient queries and updates processing. There has been great interest in storing XML data in RDBMS [1, 3, 11, 14, 15], in order to leverage the power of RDBMS for data management. However, since XML data is fundamentally different from relational data encountered in typical business applications, there are several challenges for storing XML data into relational database.

First, XML is the successor of earlier document markup languages such as SGML and HTML, primarily a document format. The implicit order among data elements, the so called document order, is important. An XML database needs a mechanism to record the relative position of data elements. Recently [15] presented how to store XML data in RDBMS preserving the document order. However, how to maintain the order upon updates is not clear.

Second, an XML database must be able to efficiently retrieve XML fragments by some XML query language, like XPath or XQuery. The edge table approach [11] treated an XML document as a tree, and generated a tuple for every XML node with its parent node identifier in the relation. To process queries with structural navigation, one self-join is needed to obtain each parent-child relationship. [1, 14] proposed to inline the information of leaf nodes into the tuple for their parents, such that the joins between a node and its leaf children are eliminated. However, to answer descendant-axis “//” or ancestor-axis in XML query, many self-joins are needed.
One popular method for maintaining the document order, which assigns ordered labels to data items, turns out to be very helpful to answer ancestor-descendant queries. Specifically, an XML document, treated as an ordered tree, is traversed in depth-first order and ordered labels are assigned to element nodes. Each node $x$ receives two numbers, the first one, $B_x$, when it is first visited, and the second one $E_x$, when it is exited. For example, Figure 1 shows an XML tree where every node is labeled by two numbers. Using this scheme, a navigation query can be converted to an interval containment test by using the following observation: for any two nodes $x$ and $y$, $x$ is an ancestor of $y$ if and only if the interval $(B_x, E_x)$ includes the interval $(B_y, E_y)$, or equivalently $B_x < B_y$ and $E_y < E_x$. Now, to answer a query “book//title” over the example, one only needs to find the nodes with tag “book” and the nodes with tag “title”, then test their labels to check the ancestor-descendant relationship. When XML data is stored in RDBMS, the ancestor-descendant queries can be processed by exactly one self-join with label comparisons as predicates, which is as efficient as child-axis. The effectiveness and efficiency of XQuery processing with the labeling scheme in comparison with other XQuery implementations is discussed in [7].

![Fig. 1. An example of an XML labeling scheme](image)

While very advantageous for queries and preserving the document order, most of the proposed labeling schemes [2][12][13][17] incur large relabeling costs. Consider the labeling scheme in Figure 1 which assigns labels from the integer domain, in sequential order. This leads to relabeling of half the nodes on average, even for a single node insertion. Alternatively, one can leave gaps in between successive labels to reduce the number of relabelings upon updates. As proved in [5], an order-preserving labeling scheme without any relabelings upon updates requires $O(n)$ bits per label, which leads to large space requirements and costly label comparisons during query processing. It is not clear how to assign the gaps between labels such that we can find a good trade-off between the number of bits used to encode the labels and the number of node relabelings each update will cause. The paper addresses this problem.

The contributions and the structure of this paper are:

1. We introduce a dynamic structure called L-Tree to maintain an order-preserving labeling scheme for XML data in the presence of updates in Section 2.
2. We analyze the amortized update cost of an L-Tree and the label size. We derive exact functions for the update complexity and the label size, and discuss how the optimal results can be achieved by choosing different tree parameters in various application settings. This is presented in Section 3.