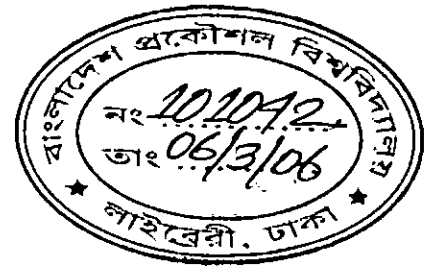


# Indexing of Extensible Markup Language Data Using a Two Dimensional Bitmap

Submitted by

**B.M. Monjurul Alom**  
Student ID: 100105019 P

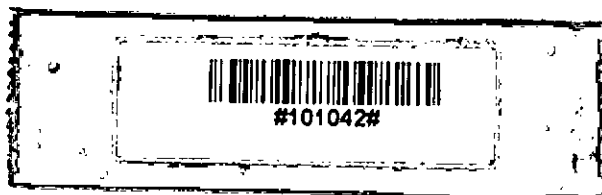
A thesis submitted to the Department of Computer Science and Engineering in partial fulfillment of the requirements for the degree of Master of Science in Engineering in Computer Science and Engineering



Supervised by

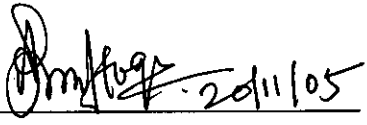

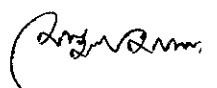

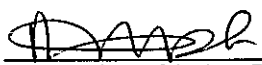
**Dr. A.S.M. Latiful Hoque**  
Associate Professor, Department of CSE, BUET.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING  
BANGLADESH UNIVERSITY OF ENGINEERING AND TECHNOLOGY, DHAKA  
NOVEMBER 2005



The thesis “**Indexing of Extensible Markup Language Data Using a Two Dimensional Bitmap**”, submitted by B.M. Monjurul Alom, Roll No. 100105019P, Session October 2001, to the Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, has been accepted as satisfactory for the partial fulfillment of the requirements for the degree of Master of Science in Engineering (Computer Science and Engineering) and approved as to its style and contents. Examination held on November 20, 2005.

**Board of Examiners**

1.   
Dr. Abu Sayed Md. Latiful Hoque  
Associate Professor  
Department of CSE  
BUET, Dhaka-1000  
Chairman  
(Supervisor)
2.   
Dr. Muhammad Masroor Ali  
Professor and Head  
Department of CSE  
BUET, Dhaka-1000  
Member  
(Ex-officio)
3.   
Dr. Md. Mostofa Akbar  
Assistant Professor  
Department of CSE  
BUET, Dhaka-1000  
Member
4.   
Dr. Md. Monirul Islam  
Associate Professor  
Department of CSE  
BUET, Dhaka-1000  
Member
5.   
Dr. Md. Mahbubur Rahman  
Associate Professor and Head  
CSE Discipline  
Khulna University  
Khulna  
Member  
(External)



# Declaration

I, hereby, declare that the work presented in this thesis is the outcome of the investigation performed by me under the supervisor of Dr. A.S.M. Latiful Hoque, Associate Professor, Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, Dhaka. I also declare that no part of this thesis and thereof has been or is being submitted elsewhere for the award of any degree or diploma

Countersigned

(Dr. A.S.M. Latiful Hoque)

Supervisor

Signature

*Malem*  
22-11-05

B.M. Monjurul Alom

## Abstract

Extensible Markup Language (XML) is a standard for representing and exchanging information on the Internet. Storing and querying of XML data has created new challenges for conventional relational database management system. This is because XML has no fixed schema, rapidly evolving, self-describing and their types are different. For these reasons, conventional indexing methods such as sparse and dense indexing, hashing and B+ trees are not satisfactory for XML data. Bitmap indexing is suitable for XML data. But the existing three-dimensional bitmap indexing method for which the space requirement is high and time consuming for searching the database.

To overcome these limitations, we have developed a *Two Dimensional Bitmap Indexing* scheme for XML data that improves the storage performance of XML data, reduce the search time to query any XML data from XML document, and improve the query performance. Our system stores XML data in a compressed form and query can be performed in this compressed representation. Storage improvement is on the average of a factor of 400:1 compared to the similar three dimensional approach. We have significant performance improvement in query processing as well.

## **Acknowledgment**

First and foremost, I would like to thank my supervisor Dr. A.S.M. Latiful Hoque, Associate Professor, Department of Computer Science and Engineering,, for his invaluable support and advice. His patience and insight to point out my mistakes forced me to become more rigorous in my reasoning. His guidance was always invaluable in all stages of my thesis work. I consider myself lucky for having the chance to work with him.

I would also like to thank Dr. Md. Masroor Ali, Professor and Head, Department of Computer Science and Engineering, BUET Dhaka, for his willingness to encourage me. I am grateful to Dr. Mahbubur Rahman, Associate Professor and Head, Computer Science and Engineering Discipline, Khulna University for his generous consent to become the external examiner with his very busy schedule. I also thank Dr. Md. Monirul Islam for his comments that have made the thesis a valuable one. I would like to thank Dr. Md. Mostafa Akbar for helping me in all aspects of my research. I am also grateful to the Alberto Mendelzon for valuable research paper.

I am also in debt with my parents, for helping me to become who I am. I dedicate this thesis to them.

<b>Contents</b>	<b>Page no</b>
Abstract	I
Acknowledgement	II
List of tables	V
List of figures	VI
<b>CHAPTER 1: INTRODUCTION</b>	<b>1</b>
1.1 Introduction	1
1.2 Overall structure of the method	2
1.3 XML Representation	3
1.3.1 Characteristics of XML Documents	3
1.3.2 Structure of XML Documents	3
1.3.3 Application of XML Documents	4
1.4 Objectives	4
1.5 Thesis Organization	5
<b>CHAPTER 2: LITERTURE SURVEY</b>	<b>6</b>
2.1 Introduction	6
2.2 Related works	6
2.3 Three Dimensional Bitmap Indexing Method	6
2.3.1 Organization	6
2.3.2 Bitcube	9
2.4 Toronto XML Indexing Method (Toxin)	10
2.5 Inverted Index	15
2.6 XML Indexing And Storage System (XISS)	20
2.7 Conclusion	23
<b>CHAPTER 3: PROPOSED 2D BITMAP INDEXING METHOD</b>	<b>24</b>
3.1 Introduction	24
3.2 Creation of Two Dimensional Matrix from XML Document	24
3.3 Construction of Bitmap	28

3.3.1 System Structure of Bitmap Indexing	31
3.4 Searching the XML Document	32
3.4.1 Querying the XML Document Database	33
3.4.2 Searching in Multiple Attributes	35
3.4.3 Example of Query with Multiple Attributes	35
3.5 Analysis of Bitmap Construction	36
3.6 Analysis of Time Complexity of the Algorithm	37
3.7 Analysis of Time Requirements for Query Operations	38
3.8 Analysis of Memory Requirements of the Algorithm	40
<b>CHAPTER 4: EXPERIMENTAL RESULTS AND DISCUSSIONS</b>	41
4.1 Introduction	41
4.2 Experimental Setup	41
4.3 Memory Requirements for Various Indexing Methods	41
4.4 Time Requirements for Various Indexing Methods	43
4.5 Word searching time	44
4.6 Word Searching Time and their Selectivity in Documents	45
4.7 Searching Time for Multiple Attributes	48
4.8 Path Construction Time	49
4.9 Discussions	50
<b>CHAPTER 5: CONCLUSIONS</b>	51
5.1 Introduction	51
5.2 Contributions	51
5.3 Future work	52
<b>REFERENCES</b>	54

## List of tables

	<b>Page no</b>
3.1 Word Dictionary	25
3.2 Path Dictionary	25
3.3 Word Searching Result	34
4.1 Dataset from XML Repository	42
4.2 Memory Needed for Various Indexing Methods	42
4.3 Time Unit Needed for Various Indexing Methods for Various Dataset	44
4.4 Word searching time	45
4.5 Word Searching Time and their Selectivity in Documents	45
4.6 Word Searching Time and their Selectivity in large Documents	47
4.7 Query in Multiple Attributes with AND Operation	48
4.8 Query in Multiple Attributes with OR Operation	49
4.9 Path Construction Time	49



## List of figures

## Page no

1.1 Overall Structure of Our Indexing Method	2
1.2 Example of XML Documents	3
2.1 XML Document	8
2.2 Example of XML Documents	8
2.3 Presence of Path in the Corresponding Documents	9
2.4 Bitcube	9
2.5 Example of BUET Database	11
2.6 XML Tree of BUET Database	12
2.7 Toxin Tree	13
2.8 Toxin Tables	13
2.9 XML Document with Numbering Process	16
2.10 Element Index	17
2.11 Text Index	17
2.12 Element Table	18
2.13 Text Table	18
2.14 Preorder Numbering Schema	21
2.15 Tables of XISS/R System	22
2.16 Indexing Structure Overview	23
3.1 Example of XML Document	24
3.2 Matrix to Get First Path	26
3.3 Matrix to Get the First Word within First Path	27
3.4 Matrix to Get the Second Word within First Path	27
3.5 Matrix to Finish the First Document	27
3.6 Two Dimensional Token-Path-Word Matrix	28
3.7 Path-Token Matrix	29
3.8 Bitmap Showing the Existence of Word	29
3.9 Division of Matrix into Block	30
3.10 Decimal Form (Bitmap)	30
3.11 System Structure of Bitmap Indexing Method	31
3.12 Flowchart of Searching Single Word	32
3.13 System Structure of Specific Query with Multiple Attributes	35

3.14 Bitmap Indexing	36
3.15 Path-Token Dictionary	37
3.16 Two Dimensional-Doc-Token-Path Matrix	37
3.17 Bitmap Matrix	37
4.1 Comparison of Memory Requirements for 3D, 2D, and Bitmap Index	43
4.2 Graph of Time Comparison for 3D, 2D and Bitmap Indexing	44
4.3 Searching Time vs Word Selectivity Relationship	46
4.4: Searching time vs words selectivity Relationship in large documents	47



# Chapter 1

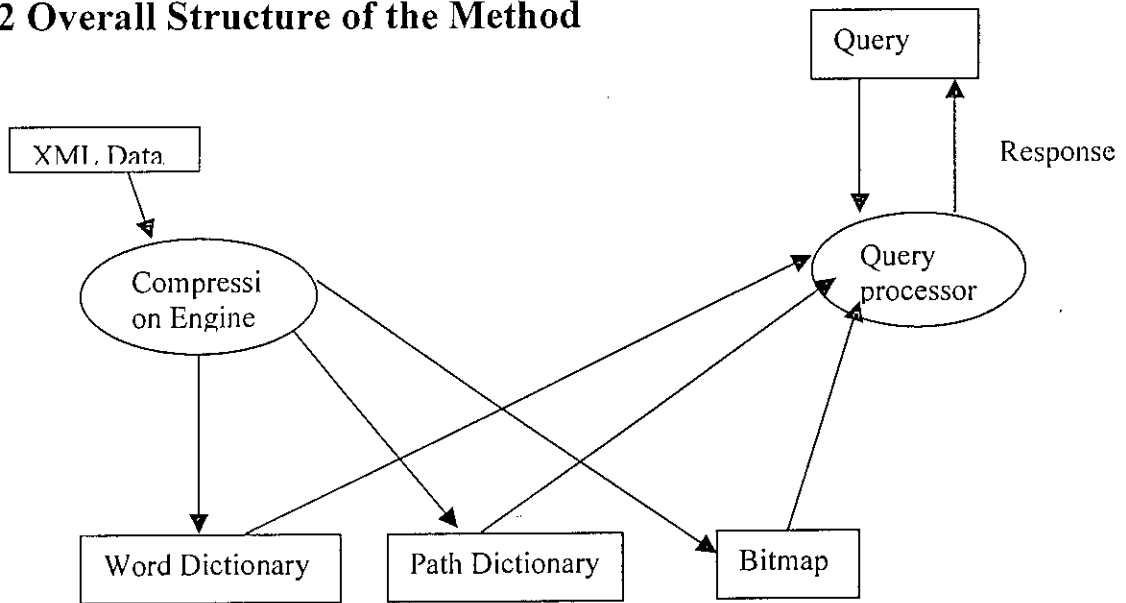
## Introduction

### 1.1 Introduction

Semistructured databases unlike traditional databases do not have a fixed schema, largely evolving, self-describing and can model heterogeneity more naturally than either relational or object-oriented databases. Example of such self-describing data is extensible markup language called XML. XML is a standard for representing and exchanging information on the Internet. Querying XML data requires an efficient indexing method. Conventional indexing methods such as Sparse and Dense indexing, Hashing, B+ trees are not satisfactory as the size of XML documents are very large and their types are different. So Bitmap indexing plays an important role for XML data. We have considered document database, each document of XML contain element path and each path contain zero, one or more words. In three-dimensional Bitmap indexing, three-dimensional matrix is required to store element-path, word and document number. In two-dimensional Bitmap indexing we require only a two-dimensional matrix, which can store element-path, existence of word and document number.

Indexing schemes for semistructured data have been developed in recent years to optimize path query processing by summarizing path information. Existing three-dimensional bitmap indexing of XML data requires large space. At the same time, querying of large XML documents database is difficult. To overcome these limitations, we have developed an indexing scheme of XML data using a two-dimensional Bitmap, providing the facility to store element-path, token and documents in a two dimensional matrix. This system contains two dictionaries; one is element-path dictionary having all the distinct element paths for all XML documents and another token dictionary containing token values for the distinct words. This indexing scheme creates a token-path-document matrix; showing the existence of XML data in specific document and in appropriate path. In this thesis we present how XML data, its path and document can be stored in a two dimensional bitmap and describe its performance over three dimension.

## 1.2 Overall Structure of the Method



**Figure 1.1: Overall structure of our Indexing Method**

Figure 1.1 represents the overall structure of our indexing method. In this indexing method XML data are taken as input from XML documents. Compression engine in which XML data are passed to create word dictionary and path dictionary. In word dictionary all the distinct words and their token numbers are stored. Path dictionary contains all the distinct element path numbers and their path contents. Compression engine which takes XML data as input and create a bitmap using path dictionary and, token dictionary. In bitmap all the decimal values of the words are stored. Query processor process query as like word searching, searching with multiple attributes using word dictionary, path dictionary and bitmap. A response is achieved by query processor to get the result of query. This system creates a new column to get a new path from XML document. To get each distinct word within that path, a new column within that path boundary is created. There is a negative sign before path number to distinguish from token value. This system will set a 1 to the corresponding document number, when any word is present in that document number. If there is path repetition among documents no new column will be created. Only the token value of the word within that path will be stored. This process continues for all XML documents and a two dimensional matrix is created. In the first row of the two-dimensional matrix contains only token value and path number. The remaining rows of the matrix represent document no and the existence of word. If any word is absent in the document, there will be a zero to the column to the corresponding document number. Two

dimensional matrix is divided into two matrixes. One is the only first row of the matrix and second matrix containing the remaining rows of the matrix. The second matrix is divided into blocks, in each block there are 16-memory cell that means 32 bytes. This system creates a decimal form that is bitmap, from the second matrix.

## 1.3 XML Representation

### 1.3.1 Characteristics of XML Data

- I. XML Representing information on the Internet.
- II. XML exchanging information on the Internet
- III. The size of the XML document is very large
- IV. Their types are different
- V. No fixed schema or rigid schema
- VI. Rapidly evolving

### 1.3.2 Structure of XML Documents

Using markup, structural information is defined in terms of *elements*, the basic components of an XML document. Each element name together with its markup delimiters is called a *tag*. The first one, which has the format `<element_name>` is called *start tag*, whereas the second has the format `</element_name>` and is called *end tag*. The string between a start tag and an end tag is called an *element content* or *value*. For instance, Title in Figure 1.2 is an element delimited by the start tag `<Title>` and the end tag `</Title>`, and its value is the string “Database Mgt System”. Attributes allow us to include any additional information within an element start tag.

<p><b>Document-1</b></p> <pre> &lt;Contacts&gt;   &lt;Contact&gt;     &lt;Name&gt;       &lt;First&gt; John Robert &lt;/First&gt;       &lt;Last&gt; Pettit &lt;/Last&gt;     &lt;/Name&gt;     &lt;Address&gt;       &lt;Street&gt; Green Road &lt;/Street&gt;       &lt;City&gt; Dhaka &lt;/City&gt;       &lt;State&gt; Dhaka &lt;/State&gt;       &lt;Zip&gt; 6200 &lt;/Zip&gt;     &lt;/Address&gt;     &lt;Tel&gt; 880-2-9256591-156 &lt;/Tel&gt;     &lt;Fax&gt; 880-2-802768 &lt;/Fax&gt;     &lt;Mobile&gt; 0176879879 &lt;/Mobile&gt;   &lt;/Contact&gt; </pre>	<p><b>Document-2</b></p> <pre> &lt;Contacts&gt;   &lt;Contact&gt;     &lt;Name&gt;       &lt;First&gt; Kelly Paul &lt;/First&gt;       &lt;last&gt; Robert &lt;/last&gt;     &lt;/Name&gt;     &lt;Address&gt;       &lt;City&gt; Dhaka &lt;/City&gt;       &lt;State&gt; Bangladesh &lt;/State&gt;     &lt;/Address&gt;     &lt;Publication&gt; Indexing of XML data Using Two dimensional Bitmap   &lt;/Publication&gt;   &lt;/Contact&gt; &lt;/Contacts&gt; </pre>	<p><b>Document-3</b></p> <pre> &lt;Contacts&gt;   &lt;Contact&gt;     &lt;Name&gt;       &lt;First&gt; Balagurusamy &lt;/Firs&gt;       &lt;last&gt; Kelly &lt;/last&gt;     &lt;/Name&gt;     &lt;Address&gt;       &lt;City&gt; Khulna &lt;/City&gt;       &lt;State&gt; Dhaka &lt;/State&gt;     &lt;/Address&gt;   &lt;/Contact&gt; &lt;/Contacts&gt; </pre>	<p><b>Document-4</b></p> <pre> &lt;Db.Main&gt;   &lt;Db&gt;     &lt;Books Info&gt;       &lt;Title&gt; Database Mgt System     &lt;/Title&gt;     &lt;Author&gt;       &lt;1<sup>st</sup>&gt; Korth &lt;/1<sup>st</sup>&gt;       &lt;2<sup>nd</sup>&gt; J.S. Martin &lt;/2<sup>nd</sup>&gt;       &lt;3<sup>rd</sup>&gt; Elmasri &lt;/3<sup>rd</sup>&gt;     &lt;/Author&gt;     &lt;Keyword&gt; SQL, Funt Dependency, Transaction, Ds System   &lt;/Keyword&gt; &lt;/Book Info&gt; &lt;Book Info&gt;   &lt;Title&gt; Information Mgt System &lt;/Title&gt;   &lt;Author&gt;     &lt;1<sup>st</sup>&gt; J.S. Martin &lt;/1<sup>st</sup>&gt;     &lt;2<sup>nd</sup>&gt; Korth &lt;/2<sup>nd</sup>&gt;   &lt;/Author&gt; &lt;/Book Info&gt; </pre>
---	---	--	---

Figure 1. 2: Example of XML document.

### 1.3.3 Application of XML Data:

- I. Using the Internet for the exchange of financial transaction information (credit card transaction, banking transaction and so on)
- II. The exchange over the Internet of medical transaction data between patients, hospitals, physicians and insurance agencies. To see a XML document that contains the full information of doctors, patients can get directions to go to doctors easily.
- III. The distribution of software via web.
- IV. Using the Internet to join the parts of distributed companies
- V. XML are reasonably clear to the user. Although it is becoming increasingly rare, and even difficult, for HTML documents to be typed manually and XML documents weren't intended to be created by human beings. XML's markup is reasonably self-explanatory
- VI. XML can be used with existing web protocols (such as HTTP and MIME) and mechanisms (such as URLs) and it does not impose any additional requirements.
- VII. XML is compatible with SGML(Standard General Mark up language) and HTML

## 1.4 Objectives

Bitmap indexing is suitable for XML data, but in a three dimensional bitmap indexing space requirement is high and time consuming for data searching. In three-dimensional Bitmap indexing, three dimensional matrix is required to store element-path, word and document number. In our proposed two-dimensional indexing requires only a two-dimensional matrix, which can store element-path, existence of word and document number.

Our objective is to:

- (i) Design a new two dimensional Bitmap index.
- (ii) Improve the storage performance of XML indexing
- (iii) Querying the XML data in compressed format
- (iv) Reduce the search time to query any XML data from XML document due to dimensionality reduction.

## **1.5 Thesis Organization**

The related work is given in chapter 2 that includes Three Dimensional Bitmap Indexing Method, Toronto XML Indexing Method (Toxin), Inverted Index, XML Indexing and Storage System (XISS).

Our proposed Two dimensional bitmap indexing method is given in chapter 3. This chapter includes creation of two dimensional matrix from XML document, construction of bitmap, flowchart of searching word, querying the bitmap, flowchart of query with multiple attributes, example of query with multiple attributes, analysis of bitmap construction, analysis of time complexity of the algorithm, analysis of AND OR query operations.

Experimental results and discussions are given in chapter 4. This chapter elaborates memory requirements for various indexing methods and their graphical representation, time requirements for various indexing methods and their graphical representation word searching time and their relationship with words selectivity. Path construction time is also given in this chapter.

Conclusion and discussion is given in chapter 5.

# Chapter 2

## Literature Survey

### 2.1 Introduction

In this chapter we have described various types of existing indexing methods. A Three-dimensional bitmap indexing, Toronto XML indexing (ToXin), XML indexing and storage system (XISS), Inverted indexing are elaborated in this chapter. Three dimensional indexing method requires large memory space, this method is time consuming for searching due to more dimensionality. ToXin can be used for both forward and backward navigation starting from any node. In this method XML database can be modeled as an edge-labeled graph. This data model carries both data (in the nodes) and schema information (in the edge). Tree traversal is not satisfactory because of forward and backward traversal of the tree. Inverted indexing supports Boolean, proximity, and ranking queries efficiently. XISS/R system includes a web based user interface, which enables stored documents to be queried via XPath. An XPath Query engine, which automatically translates XPath queries into efficient SQL statements, multiple relational schemes for comparison, reporting of performance statistics.

### 2.2 Related Work

Michel et, al. describe indexing XML data with Universal B-trees (UB-trees) [4] based on n-dimensional space. This indexing method works at the lowest level of the XML data. Another XML indexing and Querying data for regular path expressions is given in [5]. This method poses a new challenge concerning indexing and searching XML data, because conventional approaches based on tree traversal may not meet the processing requirements under heavy request. This system is based on a numbering scheme for elements. Raghav Kaushik et, al. describe updates of structure indexes of XML data in [6]. This method is based on the notion of graph bisimilarity. Updating XML data is presented in [7]. This method is based on a set of basic operation for both ordered and unordered XML data. S. Abiteboul et, al. describe Inverted indexes of XML data in [9]. This method is based on the numbering of each word and element path individually. From relation to semistructured data and XML are described



in [10]. Novel query facilities from structured documents are given in [14]. Dan Suciu et, al. elaborates Index structure for path expressions, an efficient compressor for XML data and adding structure to unstructured data in [11-13]. These methods work based on path indexing of XML data. Rizzolo et. al. describe Restructuring documents, databases and webs given in [15], that fully exploits the overall path structure of the database. Peter Buneman et, al. describes query language for unrestricted data given in [17] and elaborates query language and algebra for semistructured data given in [19]. In [18-23] it is described for query optimization of ordered and unordered semistructured data.

## 2.3 Three Dimensional Bitmap Indexing Method

A Three-dimensional bitmap indexing, for XML is presented in [1]. This method is based on element path calculation from root to each element, of each document. This system considers a document database  $D$ . Each document  $D$  is represented in XML. So,  $D$  contains XML-elements  $p$ , where  $p$  has zero or more terms  $w$  bound to it. Typical indexing requires a frequency table that is a two-dimensional matrix indicating the number of occurrence of the terms used in documents. By generalizing this idea, this system uses a three-dimensional matrix that consists of  $(d, p, w)$ .

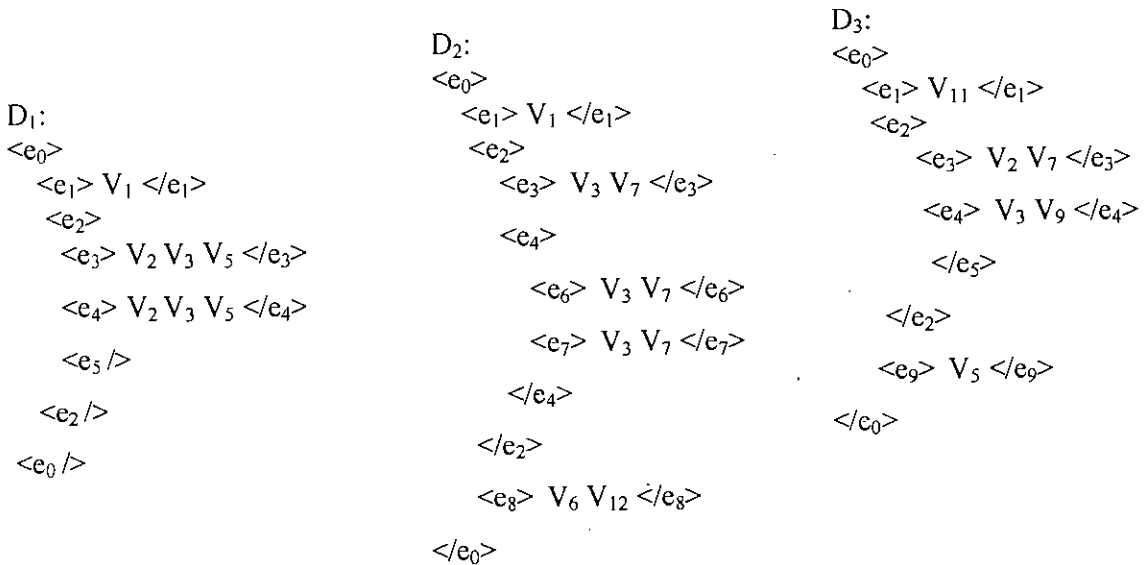
### 2.3.1 Organization

**Element Path:** Element Path, called “ePath,” is a sequence of nested elements where the most nested element is simple content element. For example, in Figure 2.1, Section subsection. Figure is an ePath, but section itself is not an ePath due to the top element `<Section>` does not have simple content.

**Element Content:** An XML-element contains (1) simple content, (2) element content, (3) empty content, or (4) reference content. As an example, consider an XML document as shown in Figure 2.1. The element `<presection>` in line (9) has a simple content. The element `<Section>` in line (1) has element content, meaning that it contains section, subsections, presection as shown in lines (2) , (4) and (9) respectively. The element `<verticalskip>` contains empty content. In line (3) there is a attribute source and an entity that representing another file “foot.gif”.

- (1) <Section>
- (2) <section> XML is originated from </section>
- (3) <footer sources="foot.gif" />
- (4) <subsection>
- (5) <figure> http://www.a.b.c/syntax.xml </figure>
- (6) <caption> XML Syntax </caption>
- (7) <verticalskip />
- (8) </subsection>
- (9) <presection> SGML was invented </presection>
- (10) </Section>

**Figure 2.1: XML Document**



**Figure 2.2: Example of XML Documents**

In Figure 2.2 is a set of simple XML documents. First, it is needed to define ePaths as follows:  $p_0=e_0.e_1$ ,  $p_1=e_0.e_2.e_3$ ,  $p_2=e_0.e_2.e_4$ ,  $p_3=e_0.e_5$ ,  $p_4=e_0.e_2.e_4.e_6$ ,  $p_5=e_0.e_2.e_4.e_7$ ,  $p_6=e_0.e_8$ ,  $p_7=e_0.e_9$ ,  $V_i$  is a(key) word that is chosen from simple content to be used for search. Now, this system constructs a bitmap index. If a document has ePath, then set the corresponding bit to 1. Otherwise, all bits are set to 0. For each ePath, documents can be represented as shown in Figure 2.3.

	P <sub>0</sub>	P <sub>1</sub>	P <sub>2</sub>	P <sub>3</sub>	P <sub>4</sub>	P <sub>5</sub>	P <sub>6</sub>	P <sub>7</sub>
D <sub>1</sub>	1	1	1	1	0	0	0	0
D <sub>2</sub>	1	1	1	0	1	1	1	0
D <sub>3</sub>	1	1	1	1	0	0	0	1

Figure 2.3: Presence of path in the corresponding documents

### 2.3.2 BitCube

XML document is defined as a set of  $(p, v)$  pairs, where  $p$  denotes an element path (or ePath) described from the root element and  $v$  denotes a word or a content for an ePath. Typical methods of handling text-based documents use a frequency table or a inverted (or signature) file that represents words for documents. However, since XML documents are represented by XML elements (or XML tags), the typical methods are not sufficient. A BitCube for XML documents is defined as  $\text{BitCube} = (d, p, v, b)$ , where  $d$  denotes XML document,  $p$  denotes ePath,  $v$  denotes word or content for ePath, and  $b$  denotes 0 or 1, the value for a bit in BitCube (if ePath contains a word, the bit is set to 1, and 0 otherwise).

A BitCube for a set of documents:  $\{d_1, d_2, d_3, d_4, d_5\}$ . Each documents  $d_1 = \{(p_0, v_1), (p_1, v_2), (p_1, v_3), (p_1, v_5), (p_2, v_3), (p_2, v_8)\}, \dots, d_3 = \{(p_0, v_{11}), (p_1, v_2), (p_1, v_7), (p_2, v_3), (p_2, v_9), \dots, (p_i, v_{i2}), (p_i, v_{i3}), (p_i, v_{i4}), \dots, (p_i, v_{ij})\}$ , and so on. An example of Bitcube is given in Figure 2.4. The approximate size of the BitCube is  $(\text{docs} * \text{words} * \text{paths}) / 8$  bytes, where docs being the number of documents that are indexed, and paths in the chosen documents.

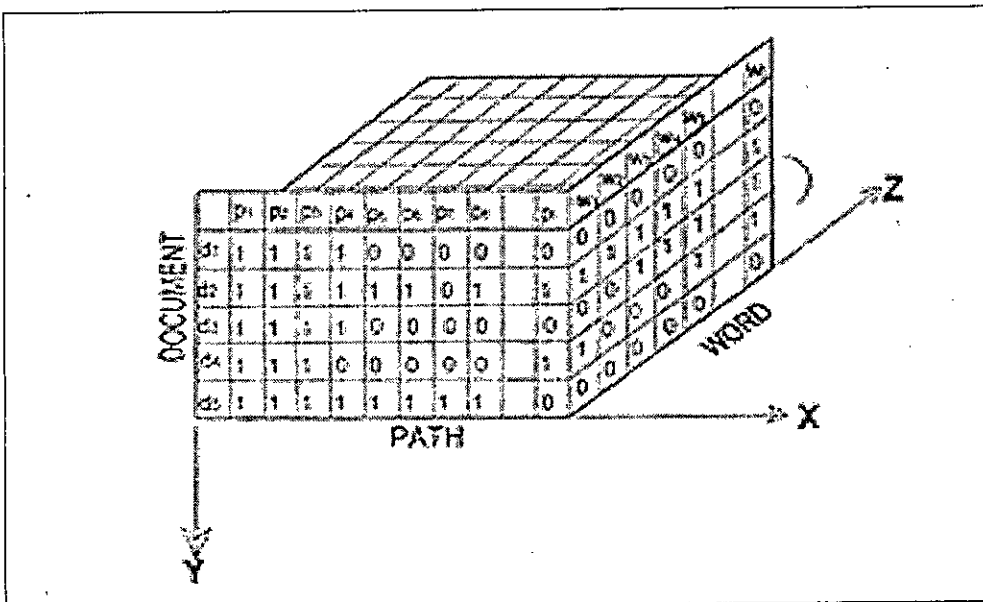


Figure 2.4: Bitcube [1]

### 2.3.3 Critical Issues of Three Dimensional Bitmap Indexing

- In three dimensional indexing documents are stored in row wise
- Element paths are stored in column wise
- In another dimension that means in Z dimension, word information are stored
- As this indexing method is three dimensional, large memory space is required
- Also this indexing method is time consuming for searching due to more dimensionality.

## 2.4 Toronto XML Indexing (ToXin)

Rizzolo et. al. describe indexing XML data with Toronto XML indexing (ToXin) in [2], that fully exploits the overall path structure of the database in all query-processing stages based on tree traversal. Most of the indexing schemes can only be applied to some query processing stages whereas others only support a limited class of queries. ToXin fully exploits the overall path structure of the database in all query-processing stages. It can be used for both forward and backward navigation starting from any node. Support navigation of the XML graph to answer any regular pat query.

### 2.4.1 System Description:

ToXin consists of two different types of index structure Value index and Path index.

Value index consists of a set of value relations that store the XML nodes and values corresponding to an index edge. A value relation is created for each edge in the index scheme that corresponds to a set of XML nodes containing values. Path index has two components; index tree and a set of instance functions.

Index Tree:

- For each edge, check whether the corresponding index edge has already been added and adds it if it was not.
- It performs a depth-first traversal of the XML tree.
- Update the instance function for the current index edge by adding the pair (parent node, child node).

A set of instance functions:

- One for each edge in the index tree.
- Keep track of parent-child relationship.
- Each instance function is stored into two redundant hash tables: forward and backward instance tables.

**Construction of XML tree:** An XML document is represented in Figure 2.5 that is represented as a XML tree in Figure 2.6. XML database can be modeled as an edge labeled graph called XML tree. This data model carries both data (in the nodes) and schema information (in the edge). Each element path is considered as a edge. Each edge has two nodes.. In Figure 2.5, BUET is the root element. In XML tree given in 2.6, BUET is considered as an edge. The value of the node of this edge is started from 1. So the value of the nodes of this edge is 1 and 2. Similarly CSE is an element path that is two times in XML document given in Figure 2.5, within root element path BUET. Two edges are created for CSE path, but their root is BUET, so nodes 2 and 3 form an edge also nodes 2 and 4 form another edge. Similarly edges have been created for sub elements of course title, courseno and year. In the leaf node of the tree all the values of the element paths are considered.

### **Construction of ToXin tree and tables:**

From the XML tree given in Figure 2.6 , ToXin tree is created given in Figure 2.7. In the XML tree those edges have element path and their leaf nodes containing element value are considered in ToXin tree as a Value table edge (VT edge). For example session element path in Figure 2.6 have values 2003-2004 and 2002-2003, in ToXin tree this edge is considered as VT1. For each VT edge one value table will be created in ToXin table given in Figure 2.8. So for session, a VT table is created that is VT1. This value table will store the value of element path and their root node. For this in VT1 node 3 and 4 is stored as it is the root node of the session 2003-2004 and session 2002-2003. Similarly for all other edges those have element Values , value table will be created. Those edges in XML tree in Figure 2.6 have only Element path will be considered in ToXin tree as a Instance Table edge (IT edge). For example BUET element path is considered in ToXin tree as IT1 edge. For each IT edge a IT table will be created that will store the parent child node for that edge. As for example for BUET element path IT1 table is crated and stores the parent node and child node 1 and 2.

Similarly for all IT edges a IT table will be created and similar types of information will be stored.

```

<BUET>
  <CSE>
    <Session> 2003-2004 </Session>
    <Course>
      <Coursetitle> DSP </Coursetitle>
      <Courseno> CSE 423 </Courseno>
      <Coursetitle> DSD </Coursetitle>
      <Courseno> CSE 467 </Courseno>
      <Courseno> CSE 461 </Courseno>
    </Course>
  </CSE>
  <CSE>
    <Session> 2002-2003 </Session>
    <Course>
      <Coursetitle> Neural </Coursetitle>
      <Courseno> CSE 433 </Courseno>
      <Coursetitle> AI </Coursetitle>
      <Courseno> CSE 477 </Courseno>
    </Course>
  </CSE>
</BUET>

```

Figure 2.5: Example of BUET Database

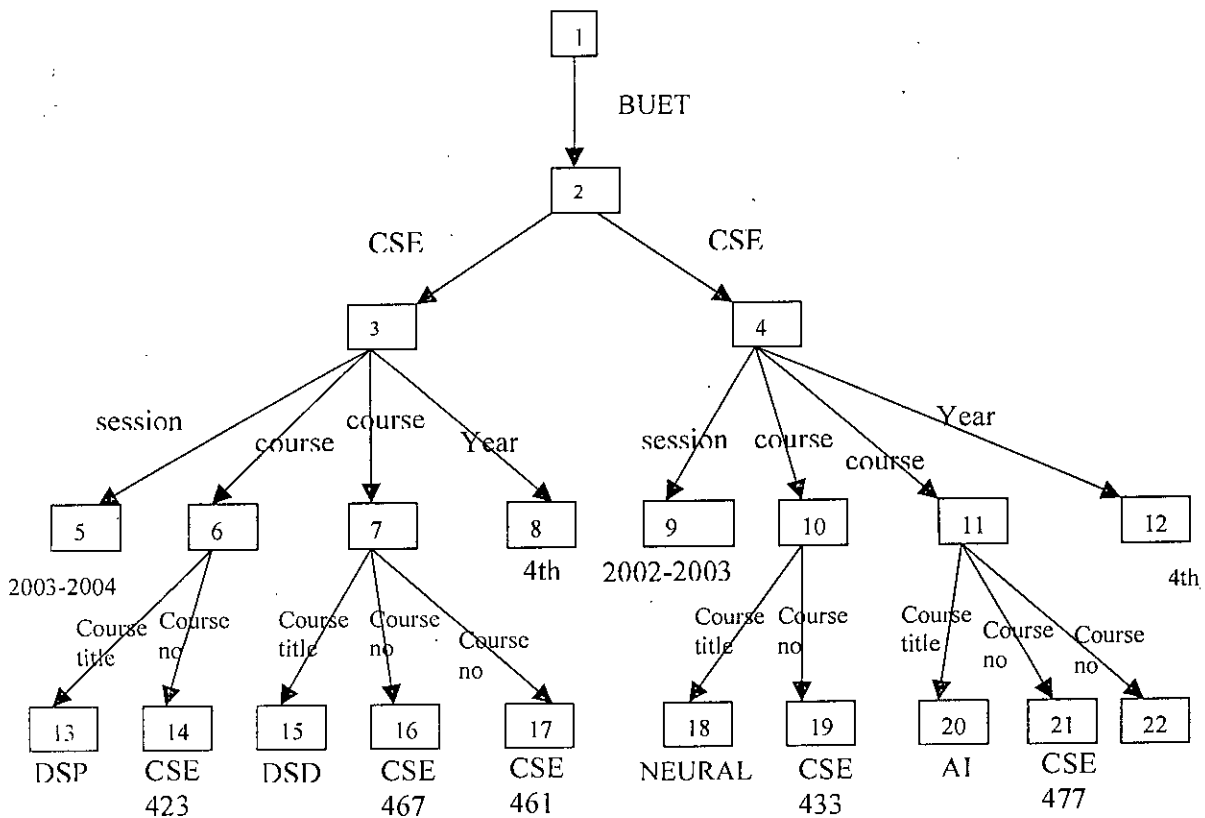


Figure 2.6: Tree of XML database given in figure 2.5

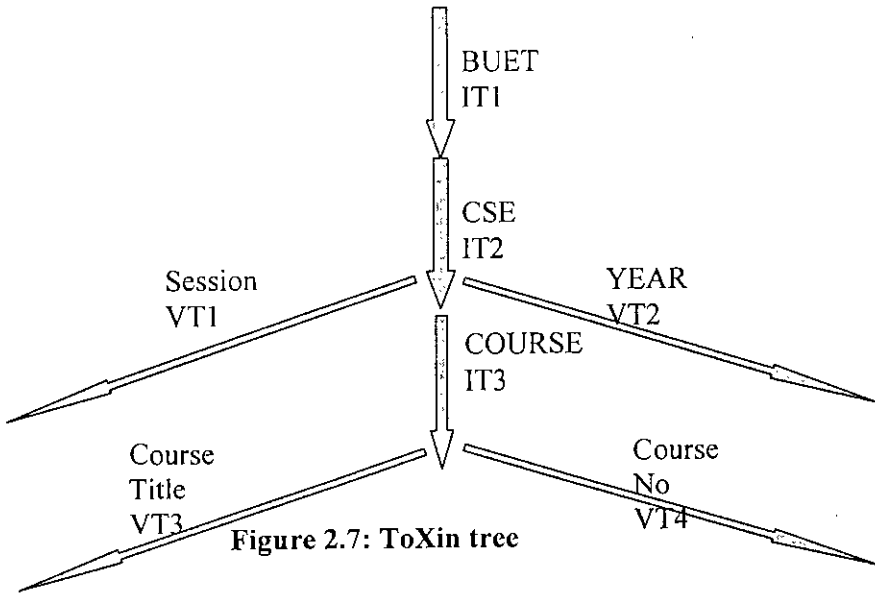


Figure 2.7: ToXin tree

VT4

Node	Value
6	"CSE-423"
7	"CSE-467"
7	"CSE-461"
10	"CSE-433"
11	"CSE-477"
11	"CSE-471"

VT1

Node	Value
3	"2003-2004"
4	"2002-2003"

VT2

Node	Value
3	"4 <sup>TH</sup> "
4	"4 <sup>TH</sup> "

VT3

Node	Value
6	"DSP"
7	"DSD"
10	"NEURAL"
11	"AI"

IT1

Parent	Child
1	2

IT2

Parent	Child
2	3
2	4

IT3

Parent	Child
3	6
3	7
4	10
4	11

Figure 2.8: ToXin Tables

## 2.4.2 Query on ToXin Method

Considering the document given in Figure 2.5. Suppose it is required to find the word AI. In this system for each element path, which has element value, a value table is created. The value table is numbered serially. The values of element paths are stored in this value table. The root node of this element path is also stored in value table. AI is a value of element path course name. VT3 is created for course name. Then this system checks the Value Table "VT3" to find out the node for "AI". We see the root node for "AI" is "11". Instance table (IT table) keeps the parent child relationship of element path. To find out the Parent of the node "11" this system check the Instance Table. This node is stored in "IT3". It is clear that the Parent node for the node "11" is the node "4". In this way the parent node for the node "4" is "2" and the parent of the node "2" is the node "1" from Instance Tables IT2 and IT1 respectively. So the path for the word "AI" is Buet.CSE.course.courseno

## 2.4.3 Critical issues of the Toxin Method:

- Most of the indexing schemes can only be applied to a limited class of queries.
- ToXin fully exploits the overall path structure of the database in all query processing stages.
- ToXin can be used for both forward and backward navigation starting from any node.
- XML database can be modeled as an edge-labeled graph
- This data model carries both data (in the nodes) and schema information (in the edge)
- Tree traversal is not satisfactory because of forward and backward traversal of the tree
- For each word, this system search from root to each leaf node that is time consuming.



## 2.5 Inverted Index

Abiteboul S, Buneman P, Suci D describe inverted indexing in [9]. Inverted index consists of two kinds of index structure. These are constructed in the following way:

**T-index:** Text words are indexed in a T-index similar to that used in a traditional IR system.

T-index consists of following element:

1. Document Number (doc-no).
2. Word Number (word-no).
3. Level Number (level).

**E-index:** Elements are indexed in an E-index, which maps elements to inverted lists. E-index consists of following element:

1. Document Number (doc-no)
2. Begin
3. End
4. Level Number (level)

### 2.5.1 Numbering Process

This system gives a number for each element or word sequentially. Consider the example given in 2.8, the numbering is started from 1 and finished with 37. From the given Figure in 2.8 this system prepares the following two Element index and Text index given in Figure 2.9 and 2.10.

```

<DUET>
  2
  <student>
    3
    <Name>
      4      5      6      7
      <First> Mahiuddin Maruf </First>
      8      9      10
      <Last> Sabbir </Last>
    11
    </Name>
    12
    <Address>
      13      14      15      16
      <Street> Jubilee Road </Street>
      17      18      19
      <City> Gazipur </City>
      20      21      22
      <Division> Dhaka </Division>
      23      24      25
      <Zip> 1700 </Zip>
      26
    </Address>
      27      28      29
      <Tel> 880-2-9256591-156 </Tel>
      30      31      32
      <Fax> 880-2-802768 </Fax>
      33      34      35
      <Mobile> 0176879879 </Mobile>
    36
  </student> </DUET>
  37

```

Figure 2.9: An XML document with numbering process

**Element index:**

<DUET> → (1,1:37,0)  
 < student> → (1,2:36,1)  
 <Name> → (1,3:11,2)  
 <First> → (1,4:7,3)  
 <Last> → (1,8:10, 3)  
 <Address> → (1, 12:26, 2 )  
 <Street> → (1, 13:16, 3 )  
 <City> → ( 1, 17:19, 3 )  
 <Division>→ (1, 20:22, 3 )  
 <Zip> → (1, 23:25, 3)  
 <Tel> → ( 1, 27:29, 2 )  
 <Fax> → (1, 30:32, 2)  
 <Mobile> → ( 1, 33: 35, 2)

**Text Index :**

Mahiuddin → (1,5,4)  
 Maruf → (1,6,4)  
 Sabbir → (1,9,4)  
 Green → (1,14,4)  
 Road → (1,15,4)  
 Gazipur → (1,18,4)  
 Dhaka → (1,21,4)  
 1700 → (1,24,4)  
 880-2-9256291 → (1, 28, 3)  
 880-2-802768 → (1, 31, 3)  
 0176879879 → (1, 34, 3)

**Figure 2.11: Text Index****Figure 2.10: Element Index**

From Figure 2.9, DUET (1, 1:37,0) represents that 1 is the document number next 1 is value of that word 37 is ending value of this document, 0 means level of that word in the document. Similarly for Mahiuddin → (1,5,4) in case of text index, 1 is the document number next 5 is value of that word, 4 is the level of that word.

**2.5.2 Relational Schema to Store Inverted Index**

- E-index and T-index can be mapped into the following two relations.
  - ELEMENTS (docno, begin, end, level)
  - TEXTS (term, docno, wordno, level)

*ELEMENT* table stores occurrences of text words. Each occurrence is stored as a table row. Two tables given in Figure 2.12 and 2.13 are created from Figure 2.10 and 2.11 respectively.

Term	Doc No	Begin	End	Level
Students	1	1	37	0
student	1	2	36	1
Name	1	3	11	2
First	1	4	7	3
Last	1	8	10	3
Address	1	12	26	2
Street	1	13	16	3
City	1	17	19	3
Division	1	20	22	3
Zip	1	23	25	3
Tel	1	27	29	2
Fax	1	30	32	2
Mobile	1	33	35	2

Figure 2.12: Element table

Term	Doc no	Word No	Level
Sabbir	1	5	4
Ahmed	1	6	4
Sabbir	1	9	4
Jubilee	1	14	4
Road	1	15	4
Khulna	1	18	4
Dhaka	1	21	4
6200	1	24	4
880-2-9256291	1	28	3
880-2-802768	1	31	3
0176879879	1	34	3

Figure 2.13: Text table

### 2.5.3 Query Process in Inverted Index Method

Suppose it is required to find the word Dhaka. We have considered the Figure given in 2.9. This system finds the word “Dhaka” from the Text-table.

For the word “Dhaka”

Doc-no = 1

Word-no = 21

Level = 4

Since the level of word is always define by the (level of element + 1).

Now level 3 of element is checked to find out the Begin and End number of the element and corresponding document number. The value 21 is between 20 and 22.

Doc-no = 1

Begin = 20

End= 22

Level = 3

### 2.5.4 Critical Issues of Inverted Index Method

- Inverted list is well suited to containment queries. It supports Boolean, proximity, and ranking queries efficiently.
- Classic inverted index data structure that maps a text word list, which numerates documents containing the word and its position within each document.
- Text words are indexed in a T-index similar to that used in a traditional IR system
- Element are indexed in an E-index, which maps elements to inverted lists.
- Elements table stores occurrences of XML elements
- Text tables stores occurrences of text words
- Each occurrence is stored as a table row.

## 2.6 XISS/R: XML Indexing and Storage System Using RDBMS

Philip J Harding, Li Q, Moon B., describe “XISS/R: XML indexing and storage System Using RDBMS.” is presented in [3]. XISS/R system based on the XISS extended preorder numbering scheme, which captures the nesting structure of XML data and provides the opportunity for storage and query processing independent of the particular structure of the data. The system includes a web base user interface, which enables stored documents to be queried via Xpath. The user interface utilizes the xpath query engine, which automatically translates Xpath queries into efficient SQL statements. So the features of the XISS/R system include:

- A web based user interface, which enables stored documents to be queried via XPath.
- An Xpath Query engine, which automatically translates XPath queries into efficient SQL statements.
- Multiple relational schemes for comparison.
- Reporting of performance statistics.

### 2.6.1 System Description

The XISS/R system consists of three components:

1. A mapping of XML data to relational schema, which is accomplished by using the extended preorder numbering scheme.
2. An Xpath query engine, allows Xpath queries to be issued on the relational implementation of the mapping of XML data.
3. A web-based user interface.

### 2.6.2 The Extended Preorder Numbering Scheme

The extended preorder numbering scheme associates each node in an XML document with a pair of numbers, the extended preorder and the range of descendents ( $\langle \text{order}, \text{size} \rangle$ , which should satisfy the following condition:

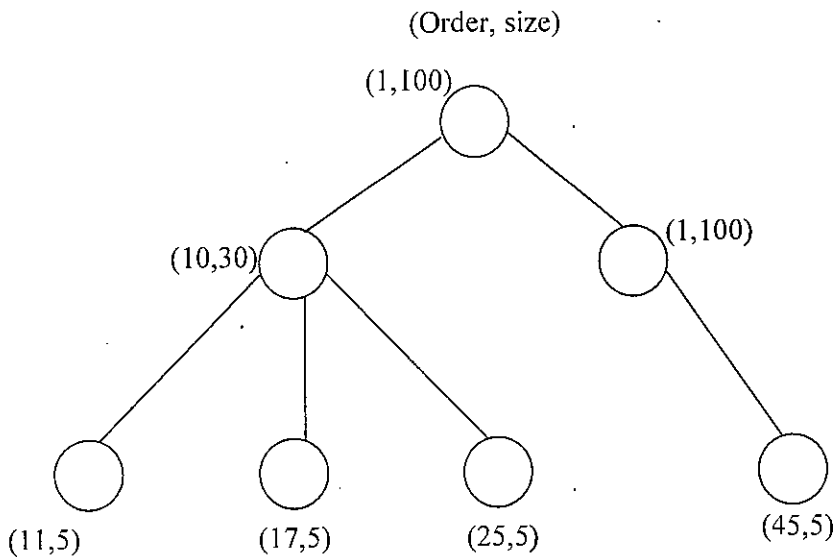
- For a tree node  $y$  and its parent  $x$ ,  $\text{order}(x) < \text{order}(y)$  and  $\text{order}(y) + \text{size}(y) \leq \text{order}(x) + \text{size}(x)$ .
- For two sibling nodes  $x$  and  $y$ , if  $x$  is the predecessor of  $y$  in preorder traversal,  $\text{order}(x) + \text{size}(x) < \text{order}(y)$ .

- Both elements and attributes use the order of the  $\langle \text{order}, \text{size} \rangle$  pair as their unique identifier in the document tree.

For a tree node  $x$ ,  $\text{size}(x)$  can be an arbitrary integer larger than the total number of the current descendents of  $x$ . This allows future insertions to be accommodating gracefully. The ancestor- descendent relationship can be determined in constant time by examining three pairs of numbers. That is, for two given nodes  $x$  and  $y$  of a tree  $T$ ,  $x$  is an ancestor of  $y$  if and only if  $\text{order}(x) < \text{order}(y) \leq \text{order}(x) + \text{size}(x)$ .

1. The Document Table consists of the Name of a document and a unique numerical Document ID .
2. The Element Table stores all element nodes.
3. The Attribute Table stores attribute nodes. The Value stores the attribute value.
4. The Text Table stores text nodes (not text values)

Within the system, Value stores the actual text. In this schema, a Document Table is a simple way to separate the document name from the element, attribute, and text relations. The element, attribute and text relations store a reference to the numerical ID of the document for each node. In the Element, Attribute, and Text tables, Order and Document ID uniquely identify any node within the system. Since all attribute nodes have a corresponding text value (or empty) string, this value is stored with the attribute node, further reducing query time.



**Figure 2.14: Preorder numbering scheme**

Element table-1	Attribute table-1	TextTable	Document Table
<b>Doc_id</b> Order Size Tag_Name Depth Child_id Next_id Att_id	<b>Doc_id</b> Order Tag_Name Depth Parent_id Next_id Att_id	<b>Doc_id</b> Order Tag_Name Depth Parent_id Next_id Att_id	<b>Doc_id</b> Name

Figure 2.15: Tables of XISS/R system (Primary keys in bold)

### 2.6.3 Relational Schema

The numbering scheme provides a unified way to store the structural relationship of XML data. However, there are a number of options for storing other necessary data from XML documents alongside such structure data. We investigate several key issues that can affect the storage and query performance:

- \* How to store element and attribute nodes.
- \* How to store tag name values.
- \* How to store value string information for text and attribute nodes.
- \* for different scheme, what kind of indexes are needed.

XISS/R requires five pieces of information for each node stored in the system, they are doc\_ID, order and size of a node in the numbering scheme, depth of a node in document tree, tag-name and text value of a node

XISS/R divides nodes into three categories, element, attribute and text.

1. The Document Table consists of the Name of a document and a unique numerical Doc\_ID.
2. The Element Table stores all element nodes.
3. The attribute table stores attribute nodes. The Value stores the attribute value.
4. The Text Table stores text nodes within the system. Value stores the actual text.



## 2.6.4 The Index Structure and Data Organization

There are three major components in the Index Structure and Data Organization. These are:

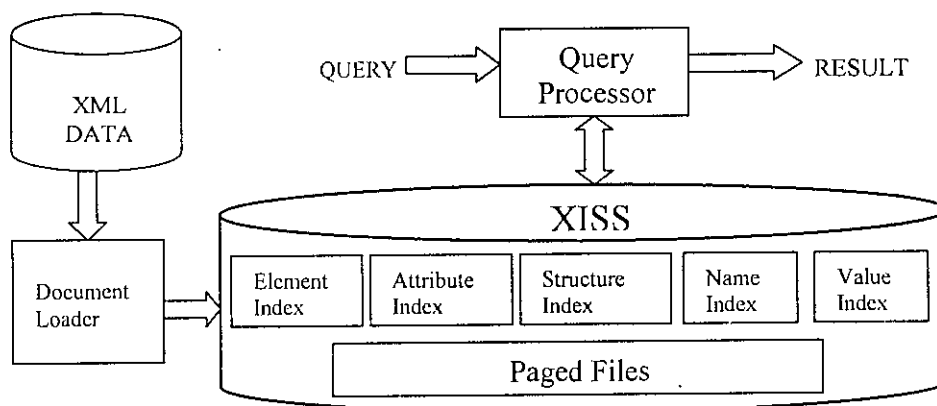
1. Element index
2. Attribute index
3. Structure index.

Two other components are name index and value table.

**Name index:** all distinct name strings are collected in the name index (identified by nid).

**Value table:** all string values (i.e. attribute value and text value) are collected in value table.

**Document Identifier:** Each XML document is also assigned a unique document identifier (did).



**Figure 2.16: Indexing Structure Overview**

## 2.7 Conclusion

In this chapter, various methods of existing indexing schemes are described. For each method query system, the system structure and the critical issues are described. In three dimensional bitmap indexing large memory space is required. Also this three dimensional indexing method is time consuming for searching due to more dimensionality. In ToXin method, tree traversal is not satisfactory because of forward and backward traversal of the tree. In ToXin it is required to search from root to each leaf node that is time consuming. In inverted indexing, inverted list is well suited to containment queries. It supports Boolean, proximity, and ranking queries efficiently.

## Proposed Two Dimensional Bitmap Indexing Method

### 3.1 Introduction

In three-dimensional Bitmap indexing, three-dimensional matrix is required to store element-path, word and document number. In two-dimensional indexing we require only a two-dimensional matrix, which can store element-path, existence of word and document number. This system creates a new column to get a new path from XML document. To get each distinct word within that path, a new column within that path boundary is created. There is a negative sign before path number to distinguish from token value. This system will set a 1 to the corresponding document number, when any word is present in that document number. If there is path repetition among documents no new column will be created. Only the token value of the word within that path will be stored. This process continues for all XML documents and a two dimensional matrix is created

### 3.2 Creation of Two-Dimensional Matrix from XML Document

To describe the creation of two-dimensional matrix from XML document we have considered XML documents given in Figure 3.1.

<p><b>Document-1</b></p> <pre>&lt;Contacts&gt; &lt;Contact&gt; &lt;Name&gt; &lt;First&gt; John Robert &lt;/First&gt; &lt;Last&gt; Pettit &lt;/Last&gt; &lt;/Name&gt; &lt;Address&gt; &lt;Street&gt; Green Road &lt;/Street&gt; &lt;City&gt; Dhaka &lt;/City&gt; &lt;State&gt; Dhaka &lt;/State&gt; &lt;Zip&gt; 6200 &lt;/Zip&gt; &lt;/Address&gt; &lt;Tel&gt; 880-2-9256591-156&lt;/Tel&gt; &lt;Fax&gt; 880-2-802768 &lt;/Fax&gt; &lt;Mobile&gt; 0176879879 &lt;/Mobile&gt; &lt;/Contact&gt; &lt;/Contacts&gt;</pre>	<p><b>Document-2</b></p> <pre>&lt;Contacts&gt; &lt;Contact&gt; &lt;Name&gt; &lt;First&gt; Kelly Paul &lt;/First&gt; &lt;last&gt; Robert &lt;/last&gt; &lt;/Name&gt; &lt;Address&gt; &lt;City&gt; Dhaka &lt;/City&gt; &lt;State&gt; Bangladesh &lt;/State&gt; &lt;/Address&gt; &lt;Publication&gt;" Indexing of XML data Using Two dimensional Bitmap &lt;/Publication&gt; &lt;/Contact&gt; &lt;/Contacts&gt;</pre>	<p><b>Document-3</b></p> <pre>&lt;Contacts&gt; &lt;Contact&gt; &lt;Name&gt; &lt;First&gt; Balagurusamy&lt;/First&gt; &lt;last&gt; Kelly &lt;/last&gt; &lt;Name&gt; &lt;Address&gt; &lt;City&gt; Khulna &lt;/City&gt; &lt;State&gt; Dhaka &lt;/State&gt; &lt;Address&gt; &lt;/Contact&gt; &lt;/Contacts&gt;</pre>	<p><b>Document-4</b></p> <pre>&lt;Db.Main&gt; &lt;Db&gt; &lt;Books Info&gt; &lt;Title&gt; Database Mgt System &lt;/Title&gt; &lt;Author&gt; &lt;1<sup>st</sup>&gt; Korth &lt;/1<sup>st</sup>&gt; &lt;2<sup>nd</sup>&gt; J.S. Martin &lt;/2<sup>nd</sup>&gt; &lt;3<sup>rd</sup>&gt; Elmasri &lt;/3<sup>rd</sup>&gt; &lt;/Author&gt; &lt;Keyword&gt; SQL, Funt Dependency, Transaction, Ds System &lt;/Keyword&gt; &lt;/Book Info&gt; &lt;Book Info&gt; &lt;Title&gt; Information Mgt System &lt;/Title&gt; &lt;Author&gt; &lt;1<sup>st</sup>&gt; J.S. Martin &lt;/1<sup>st</sup>&gt; &lt;2<sup>nd</sup>&gt; Korth &lt;/2<sup>nd</sup>&gt; &lt;/Author&gt; &lt;/Book Info&gt; &lt;/Db&gt; &lt;/Db.Main&gt;</pre>
--	--	--	--

Figure 3.1: Example of XML documents.

For each distinct word from XML documents given in Figure 3.1, this system creates a token-number that is stored in token dictionary and the corresponding distinct word is also stored in token dictionary. Path-number are also created in this system serially for all distinct paths from XML documents. This path-number and its contents is stored in element-path dictionary. Token dictionary and element-path dictionary are given in Table 3.1 and in Table 3.2.

**Table 3.1: Word dictionary**

Token-number      Words

1	John
2	Robert
3	Pettit
4	Green
5	Road
6	Dhaka
7	6200
8	880-2-9256591-156
9	880-2-802768
10	0176879879
11	Kelly
12	Paul
13	Bangladesh
14	Indexing
15	Of
16	XML
17	data
18	Using
19	Two
20	dimensional
21	Bitmap
22	Balagurusamy
23	Khulna
24	Database
25	Mgt
26	System
27	Korth
28	J.S.
29	Martin
30	Elmasri
31	SQL
32	Funt
33	Dependency
34	Transaction
35	Ds
36	Information

**Table 3.2: Element-path dictionary**

Path-number      Contents

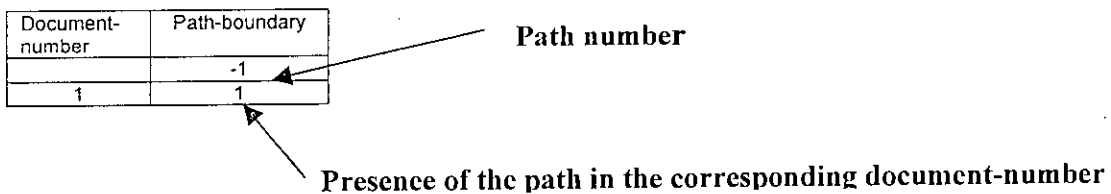
1	Contacts.contact.Name.First
2	Contacts.contact.Name.Last
3	Contacts.contact.Address.Street
4	Contacts.contact.Address.City
5	Contacts.contact.Address.State
6	Contacts.contact.Address.Zip
7	Contacts.contact.Tel
8	Contacts.contact.Fax
9	Contacts.contact.Mobile
10	Contacts.contact.Publication
11	Db.Bookinfo.Title
12	Db.Bookinfo.Author.1st
13	Db.Bookinfo.Author.2nd
14	Db.Bookinfo.Author.3rd
15	Db.Bookinfo.Keyword

Considering Document-1, Document-2, Document-3 and Document-4 given in Figure-3.1, Contacts.contact.Name.First path is created and its path number is assigned 1 and it is stored in Table 3.2. Similarly all the paths are created and their path numbers are assigned and stored in path dictionary in Table-3.2. For word John , 1 is assigned its token value and all the

tokens are created for all the distinct words serially and stored in token dictionary given in table 3.1. Using Table-3.1 and Table-3.2, this system creates the following two dimensional matrix given in Figure 3.2 step by step. This method can be implemented for any XML documents. The first column is created only for document-number. It has two rows. In second row there is a 1 represents the document number 1. To get each distinct path-number from documents a new column is created for that path. This system set a negative sign in the first row of the column, before the path-number and set a 1 in the second row of that column. The 1 in the second row represents the path is present to the corresponding document number. To get each distinct word from XML documents this system creates a new column for that word within the path boundary. Similarly a 1 is set in the second row of the column, token number is assigned to the first row of the column within that path boundary.

**Step 1:**

After getting the first path from Document, a matrix is created which has two rows and two columns. In the first row of the second column a 1 is assigned for the path number. Also a negative sign is assigned before the path number to distinguish from token number of the word. In the second row of the first column a 1 is assigned that represents the document number, in the second column of the second row another 1 will be assigned to represent the presence of the path number. When any new word is found within that path number a new column will be created and similarly 1 will be assigned as in before it is assigned. We have considered the Figure 3.1.



**Figure 3.2: Matrix to get first path**

**Step-2:**

After insertion of John, Token-number=1 given in Table-1, from Document-1 given in Figure-3.1, the following matrix will be created. Here the second column is created from the Figure given in 3.2. The first row of the second column is the token number of the word John. In the second row of the second column a 1 is assigned to represent that the word is present in the document number 1.

Document-number	Token-number	Path-boundary
	1	-1
1	1	1

Path number

Presence of the word to the corresponding Document-number-1

**Figure 3.3: Matrix to get the first word within first path**

**Step- 3:**

Similarly after insertion of Robert, Token-number=2 given in Table-1, from Document-number-1 given in Figure3.1

	1	2	-1
1	1	1	1

Path number

Presence of path in document

Document number                      Presence of word in document number

**Figure 3.4: Matrix to get the second word within first path**

**Step- 4:** Similarly this process continues until to get all words and paths from Document-1 in Figure-3.1 and the two dimensional matrix is as follows:

	1	2	-1	3	-2	4	5	-3	6	-4	6	-5	7	-6	8	-7	9	-8	10	-9
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

**Figure 3.5: Matrix to finish the first document**

**Step 5:** Similarly this process continues until to get all distinct words and paths from Document-1, Document-2, Document-3 and Document-4 given in Figure-3.1 and the following two dimensional matrixes is created.

	1	2	11	12	22	-1	3	2	11	-2	4	5	-3	6	23	-4	6	13	-5	7	-6	8	-7
1	1	1	0	0	0	1	1	0	0	1	1	1	1	1	0	1	1	0	1	1	1	1	1
2	0	0	1	1	0	1	0	1	0	1	0	0	0	1	0	1	0	1	1	0	0	0	0
3	0	0	0	0	1	1	0	0	1	1	0	0	0	0	1	1	1	0	1	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

9	-8	10	-9	14	15	16	17	18	19	20	21	-10	24	25	26	-11
1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	1	1	1	1	1	1	1	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1

27	28	29	-12	28	29	27	-13	30	-14	31	32	33	34	35	26	-15
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

**Figure 3.6: Two dimensional Token-Path-Word Matrix**

### 3.3 Construction of Bitmap

The two dimensional Token-Path-Word matrix given in Figure 3.5 has got two components; path-token matrix and word existence matrix.

#### Path-Token matrix:

The first row of the matrix in Figure 3.5 is the path-token matrix; the remaining rows are the word existence matrix. In the path token matrix given in Figure 3.6 represents the token number of the word and the path number. As for example 1,2, 11, 12, 22 are within path number -1, represents that path number 1 contains the above token number of the word. From Table 3.1 it is seen that the words John, Robert, Kelly, Poul, Balagurusamy have the token number 1, 2, 11, 12, 22 and from Figure 3.1 they are within path number 1 in all documents.

#### Bitmap showing the word existence matrix:

In word existence matrix 1 represents the word is present to the corresponding document number, 0 represents the word is absent to the corresponding document number. From Figure 3.7 this system transforms the matrix into blocks given in Figure 3.8.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	.....	
1	2	11	12	22	-1	3	2	11	-2	4	5	-3	6	23	-4	6	13	-5	7	-6	8
9	-8	10	-9	14	15	16	17	18	19	20	21	-10	24	25	26	-11					
39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55					
27	28	29	-12	28	29	27	-13	30	-14	31	32	33	34	35	26	-15					

**Figure 3. 7: Path\_Token Matrix**

1	1	1	0	0	0	1	1	0	0	1	1	1	1	1	0	1	1	0	1	1	1	1	1	1	1	1	1	0	0
2	0	0	1	1	0	1	0	1	0	1	0	0	0	1	0	1	0	1	1	0	0	0	0	0	0	0	0	1	1
3	0	0	0	0	1	1	0	0	1	1	0	0	0	0	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

**Figure 3.8: Bitmap showing the existence of word in documents.**

Figure 3.3 is the first row of the matrix given in Figure 3.2. In this method we have considered document number as page number and in each block there are 16 memory cells. If there are less than 16 memory cells in a block of the matrix, this system takes the rest of the cell in a block containing all zero. So in each block there are 16 offset addresses starting from 0 to 15. This system converts the value of each block into decimal form. During searching a word or path number this system calculate the followings:

Block no = the index of token value of searching word /16. The offset address = the index of token value token value of searching word %16. The path number = ABS (negative value of path boundary).

For each block, this system converts the decimal value into binary form. If there is a 1 in the corresponding offset position of that word then the corresponding document number is the document number for that word.

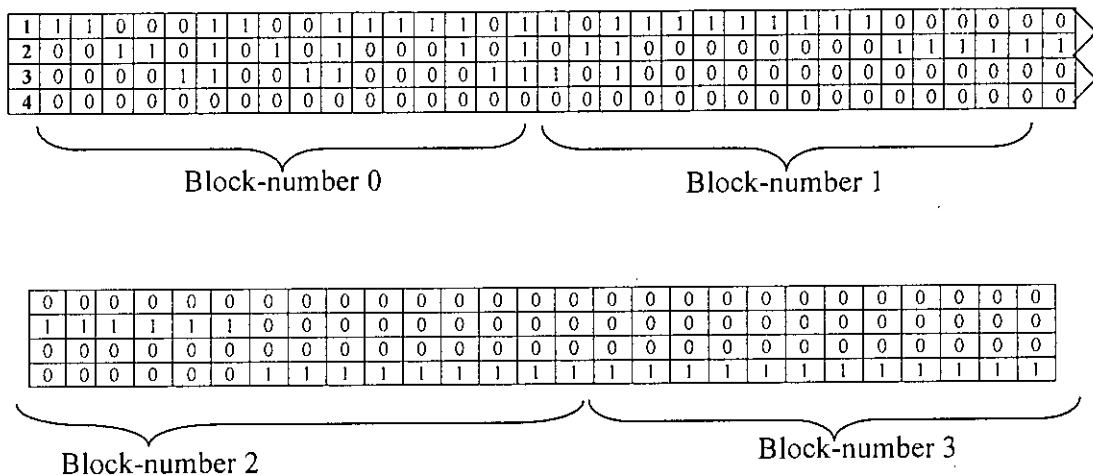


Figure 3.9: Division of matrix into blocks.

The matrix given in Figure 3.8 is decimal form that is Bitmap converted from Figure-3.9.

Document-number	Block-0	Block-1	Block-2	Block-3
Document-1	50813	49120	0	0
Document-2	13693	24607	32256	0
Document-3	3267	40960	0	0
Document-4	0	0	511	65520

Figure 3.10: Decimal form (Bitmap)

Consider the value 50813 given in Block-0, this value is converted from the binary representation of 110001100111101 from Figure 3.8. In such way all the values are converted in Figure 3.9.



### 3.3.1 System Structure of Bitmap Indexing Method

The overall system structure is given in Figure 3.10. From different XML documents, this method prepares two dictionaries, one is word dictionary and other is path dictionary. Word dictionary stores all the distinct words and their token number, path dictionary stores all the path number and path contents. From these two dictionaries and different XML documents two-dimensional token-path-word matrix is created. This matrix is then split into two matrix. One is for word existence and other is for path boundary and token value. Word existence matrix is divided into blocks; from this matrix bitmap is created.

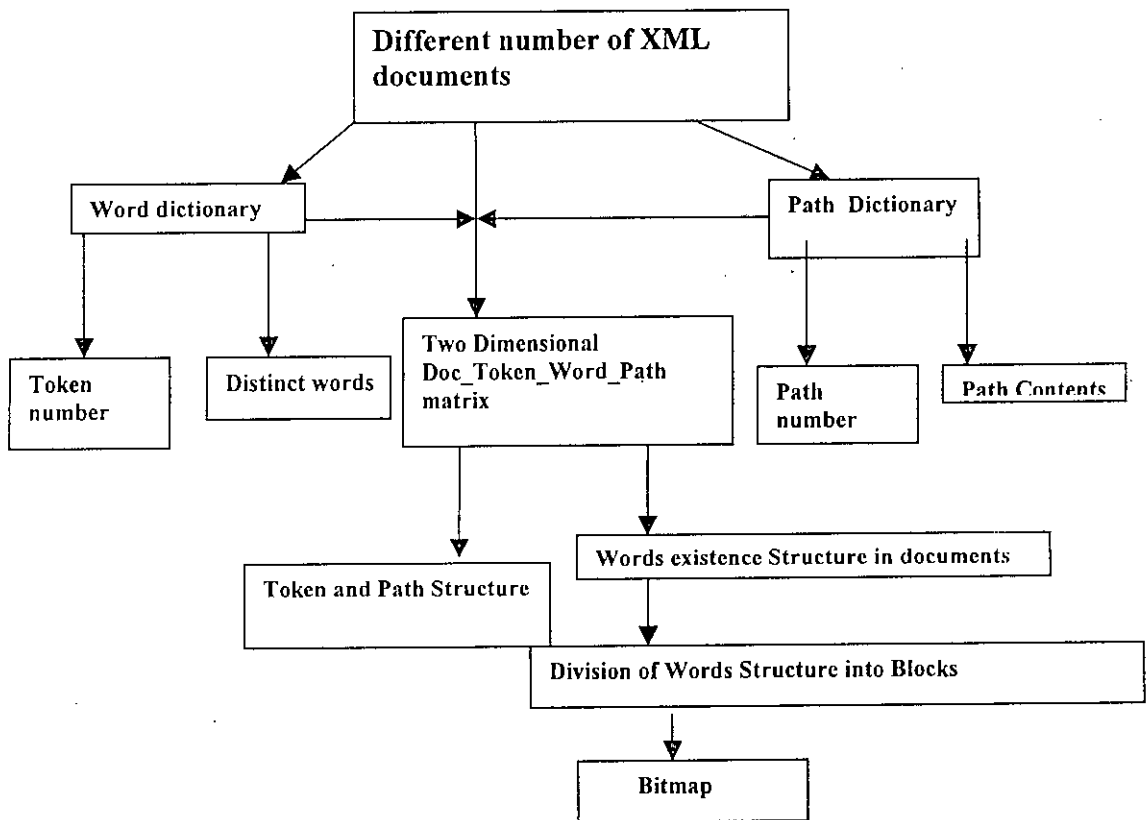


Figure 3.11: System Structure Of Bitmap Indexing Method

### 3. 4 Searching XML Documents

From all the XML documents or out of the XML documents, any word is selected to search. In this method the token number of the word from token dictionary is searched. If the token number is found, path number is searched from path\_token matrix.. Then the index of that token number is calculated from path\_token matrix. Block number is created from that index value divided by 16. Offset position is calculated from the index value % 16. This system then takes all the decimal values of bitmap matrix of the corresponding block number. Each decimal value is converted into binary form and checked for a 1 in the calculated offset position. If it is ok, the corresponding word is present in the document, otherwise it is absent.

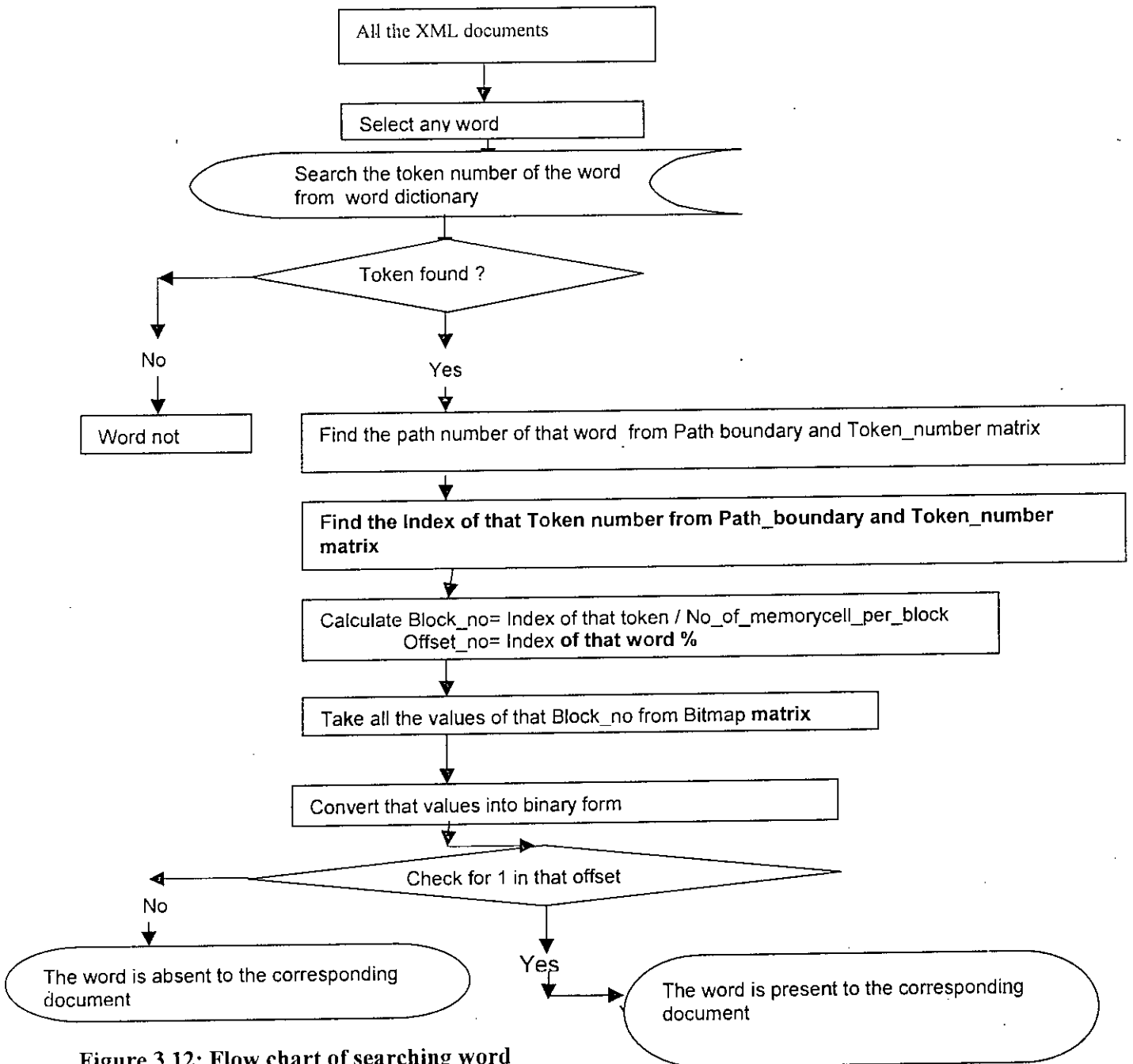


Figure 3.12: Flow chart of searching word

### 3.4.1 Querying the XML Document Database with Example

To search anything with specification is represented as query. In our method we have the following types of query are possible. To search any single word from XML documents Searching with multiple attributes in the documents.

Example of searching any word is as follows:

Suppose we want to Query Dhaka as a City from the given documents in Figure 3.1.

Select City

From Documents

Where City="Dhaka"

The token number of Dhaka = 6 from token dictionary.

Index of that Token number = 13 from Figure-3 .

Block no =  $13/16 = 0$  , Offset address=  $13 \% 16 = 13$  ,

The Token number 13 in Figure-3.2 is within path boundary 4.

In this path Dhaka is as a city. Path address =  $ABS(-4) = 4$

Decimal value of block 0, Row-1 = 50813 its binary is {1100011001111 101}

Started from 0...15 .Binary value of offset address (13) = 1, this is present in the Document -1. So, we can decide that the word "DHAKA" as a City is now in DOCUMENT-1, PATH

4. Decimal value of block 0 , Row-2(document-2) = 13693

Binary value=

0 0 1 1 0 1 0 1 0 1 0 0 0 1 0 1

Offset address (13) = 1

So, we now decide that the word "DHAKA" is in now DOCUMENT 2 , PATH 4 TOKEN NO 6.

Decimal value of block 0, Row- 3(document-3)= 3267

Binary value=

0 0 0 0 1 1 0 0 1 1 0 0 0 0 1 1

Offset address(13) = 0

So, we now decide that the word "DHAKA" is not in now DOCUMENT 3.

Decimal value of block 0,

Page 4(document-4)= 3267

Binary value=

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Offset address(13) = 0

So, we now decide that the word "DHAKA" is not in now DOCUMENT 4.

Suppose we want to Query Dhaka as a State from the given documents in Figure-1.

This system works as follows:

Select State

From Documents

Where State="Dhaka"

The token number of Dhaka = 6 from token dictionary.

Index of that Token number = 16 from Figure-3 As a state.

Block no =  $16/16 = 1$  , Offset address =  $16 \% 16 = 0$  ,

The Token number 16 in Figure-3.2 is within path boundary 5

In this path Dhaka is as a State. Path address =  $ABS(-5) = 5$

Decimal value of block 1, page 0 (means document-1) = 49120 its binary is

1 0 1 1 1 1 1 1 1 1 1 0 0 0 0 0

Started from 0...15 .Binary value of offset address (16) = 0, this is not present in the Document -1. Searching result for word dhaka in Example-1.1, using our method Similarly this system can search any word from given xml documents.

Table 3.3: Word searching result

Word_Name	Doc_no	Path_no	Path_address
	0	4	Contacts.contact.Address.City
Dhaka	1	4	Contacts.contact.Address.City
	0	5	Contacts.contact.Address.State
	2	5	Contacts.contact.Address.State

### 3.4.2 Searching in Multiple Attributes

In this method, any portion of path name is selected and the specific value of the path is selected. This method finds the path number from path dictionary and token from token dictionary. If they are found, path-no and token-no are matched into path-token matrix. In the path-token matrix, if the path-no and token-no are matched then the searching system as like word searching system given in 3.4.

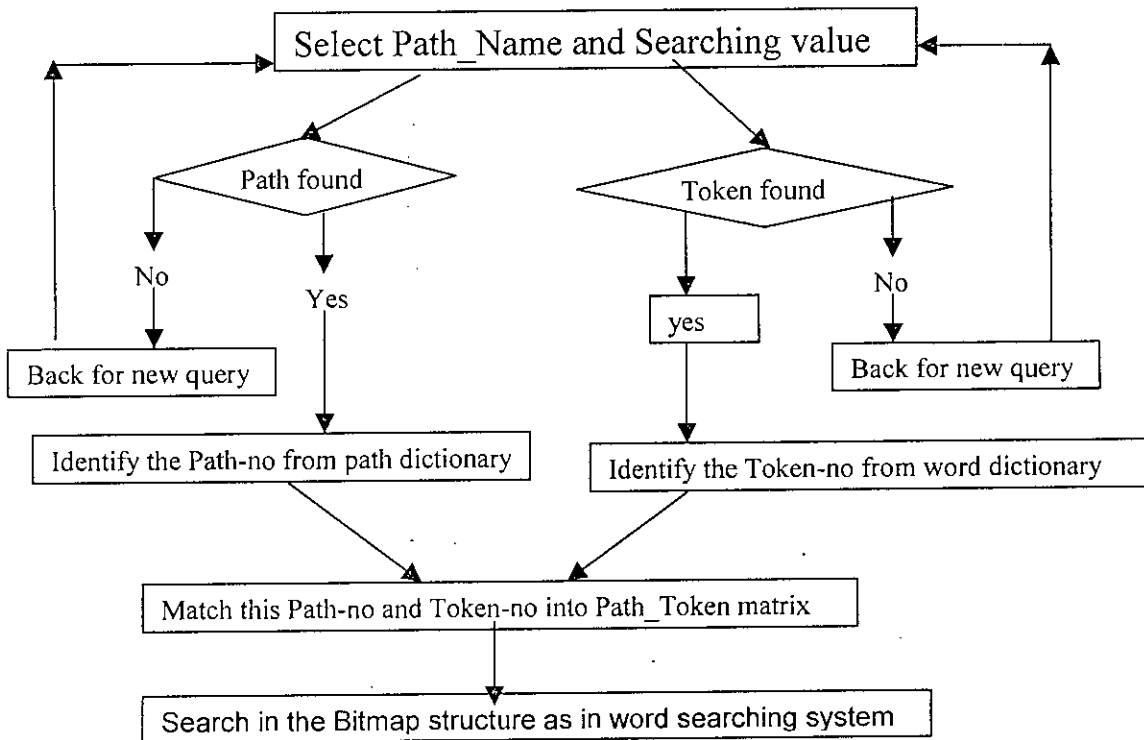


Figure 3.13: System structure of Specific Query with multiple attributes

### 3.4.3 Example of Query with Multiple Attributes:

Select Title

From documents

where 1ST author ="Korth" and 2ND author ="J.S.Martin"

This system finds the path number from path dictionary given in table-2. Path number of 1ST author and 2ND author is 12 and 13. If the path number of 1ST author is not found in path dictionary then this system will back for new query. This system also finds the token value of **Korth** from word dictionary. If it is found, then this system finds the token value of **Martin** also from word dictionary. If the word is found then this system will check either **Korth** is for 1ST author and **Martin** is for 2ND author from Path-Token matrix. The

token value of Korth is 27 and Martin is 29 from word dictionary for example given in Figure-3.1. From Figure-3.2 that is Token\_Path Matrix, we see that token 27 and 29 is under path boundary 12 and 13. Path 12 and 13 are first author and second author from path dictionary. Now this system will find in which documents these words exist. This searching system is as like the word searching system that is explained in 3.6.

### 3.5 Analysis of Bitmap Construction

We have developed an algorithm to implement the method that is given in Figure-3.14 . Path-Token\_Dictionary is used to create word and path dictionary. Each distinct word and distinct path has a separate token number and path number. The function Two\_Dimension\_Doc-Token\_Path\_Matrix is used to create the matrix that represents the word existence in documents. Bitmap matrix is the function used to create the matrix that is in compressed format. All the queries are performed in bitmap matrix.

```
Algorithm Bitmap_Indexing(){
Path-Token_Dictionary();
Two_Dimension_Doc-Token_Path_Matrix();
Bitmap matrix(); }
```

Figure 3.14: Bitmap Indexing

```
Path-Token_Dictionary(){
While(!EOF(xml_file)) do{
If (XML word) consist a start_path push this word onto a temporary path
table;
If (XML word) be a word without start & end_path then {
Push the word into token_dictionary_table with checking either exist or not;
If (!exist(token_dic[XML_word])) then {
Store the word in the token_dictionary_table;
Return the token_no for the word and store it into temp_token_table; }
Else return the token_no for the word and store it into temp_token_table; }
If (XML word) consists a end_path then {

Compare(); // this temp_path to the original path in the original_path_table
If (XML word does not exist) then { Create a new path_no for this path;
Store this path to original_path_table and return the path_no; }
Else return the only path_no for the existing path; }
If (two or more consecutive XML words) consist end_path then
pop the top element from temp_path_table; // no update is necessary
} // While } //end path_token_dictionary
```

Figure 3.15: Path-token-dictionary

```

Two_Dimension_Doc-Token_Path_Matrix(){
//first row of the matrix contain token & path_no //first column contain document no
for each new path_no do {
create a new column in doc_token_path_matrix;
store the negative value of path_no; //negative value to distinguish from token_value;
for each new token_no within this path_no do {
create a new column in doc_token_path_matrix within this path_boundary;
insert the token_no within this path_no;
insert 1 to the corresponding document ; } } }

```

Figure 3.16: Two\_Dimension\_Doc-Token\_Path\_Matrix()

```

Bitmap_Matrix(){
X=the no_of_bits_per_block from user
Index=Find_index_token(); //return the index of searching token_no from doc_token_path
matrix
Block_no=index / X;
Offset=index % X;
for each Block of doc_token_path matrix do {Convert binary to decimal;
store the decimal value into Bitmap_table; }
for each row of the Bitmap_table do { //row represent block no....column represent document.
If (the decimal value) of the block_no consist '1' in the offset position then
searching word is found to the corresponding document; }
//To get path_no search from the index of searching token, in the first row of the
doc_token_path_matrix until found a negative value, ABS(negative value) is path_no.}

```

Figure 3.17: Bitmap\_matrix

### 3.6 Analysis of Time Complexity of the Algorithm

Let us consider, W be the total no of distinct words, P be the total no of distinct path, and D be the number of documents.

Minimum time complexity of each method  $O(1)$

**In three dimensional indexing**, document is stored in one dimension, element path is stored in another dimension and word information is stored in another dimension. So, time complexity of three dimensional bitmap indexing is :  $O(W * P * D)$ .

Average time complexity of three dimensional indexing is  $O(((W * P * D) + 1) / 2)$

**In our two dimensional indexing**, the word information and path information is stored together in row wise and document information in column wise. So, time complexity of two dimensional bitmap indexing is :  $O((W + P) * D)$ .

Average time complexity of two dimensional indexing is  $O(\frac{((W+P)*D)+1}{2})$

**In our bitmap indexing** word information and path information is stored together in row wise and document information in column wise and there is a division by `no_bits_per_block`. Here we have considered 256 bits per block that means 32 bytes. So, time complexity of bitmap indexing is :  $O(\frac{(W+P)}{\text{no\_bits\_per\_block}}*D)$ .

Average time complexity of bitmap indexing is  $O(\frac{((W+P)/\text{no\_bits\_per\_block})*D+1}{2})$

### 3.7 Analysis of Time Requirements for Query Operations

We have considered 'AND' and 'OR' operation within the predicate of the query for the analysis of time requirement.

#### 3.7.1 Analysis of Time Requirements for AND Operations

Let us consider, the total time requirements for word dictionary searching is  $T_{dic}$ , the time requirement for path dictionary searching is  $T_{path}$  and the time requirements for bitmap searching is  $T_{bitmap}$ .

So total time requirements for single attribute of a single word is  $T_{q/word} = T_{dic} + T_{path} + T_{bitmap}$

$T_{q(min)} = T_{dic}$ , the time to search only the dictionary.

In case of multiple attributes where all conditions are true, time requirements is  $T_{q(max)} = n * T_{q/word}$ , where n is the total number of attributes.

In case of multiple attributes of AND operation where any condition is false. This may happen in three ways. If all the conditions are true except the last condition, if all the conditions are true except first condition, if first condition and last condition are true but any condition between first and last is false.

Time requirements (if all the conditions are true except the last condition) is

$T_{m1q} = (n-1) * T_{q/word} + T_{dic}$ , where n is the total number of attributes, here first (n-1) attributes are present but last attribute is absent. So for last attribute the time is only dictionary searching that is  $T_{dic}$ .

Time requirements (if all the conditions are true except the first condition) in case of multiple attributes is  $T_{m2q} = T_{dic} + (n-1) * T_{q/word}$  where n is the total number of attributes, here last (n-1) attributes are present but first attribute is absent. So this time is only dictionary searching that  $T_{dic}$



Time requirements (if first condition and last condition are true but any condition between first and last is false) is  $T_{m3q} = 2 * T_{q/word} + (n-2) * T_{dic}$

In case of multiple attributes of AND operation where any attribute is absent, the total time requirements is  $T_{andq} = T_{m1q} + T_{m2q} + T_{m3q}$

$$\begin{aligned}
 &= (n-1) * T_{q/word} + T_{dic} + T_{dic} + (n-1) * T_{q/word} + 2 * T_{q/word} + (n-2) * T_{dic} \\
 &= 2 * (n-1) * T_{q/word} + 2 * T_{dic} + 2 * T_{q/word} + (n-2) * T_{dic} \\
 &= 2 * n * T_{q/word} - 2 * T_{q/word} + 2 * T_{dic} + 2 * T_{q/word} + n * T_{dic} - 2 * T_{dic} \\
 &= 2 * n * T_{q/word} + n * T_{dic}
 \end{aligned}$$

Average time requirements for AND operations is:  $((n+1)/2) * T_{q/word}$

### 3.7.2 Analysis of Time Requirements for OR Operations

Let us consider, the total time requirements for word dictionary searching is  $T_{dic}$ , the time requirement for path dictionary searching is  $T_{path}$  and the time requirements for bitmap searching is  $T_{bitmap}$ .

So total time requirements for single attribute of single word is  $T_{q/word} = T_{dic} + T_{path} + T_{bitmap}$

$T_{q(min)} = T_{dic}$ , the time to search only the dictionary.

In case of multiple attributes where all the attributes are present, time requirements for OR operation is  $T_{q(max)} = n * T_{q/word}$ , where n is the total number of attributes.

In case of multiple attributes of OR operation where any attribute is absent, this may happen in three ways. If all the attributes are present except first attribute, if all the attributes are present except last attribute, if first and last attributes are present any attribute but any attribute is absent between first and last attribute.

Time requirements (if all attributes are present except the last attribute) is

$T_{lorq} = \max \{ (n-1) * T_{q/word}, T_{dic} \}$  where n is the total number of attributes, here first (n-1) attributes are present but last attribute is absent. So for the last attribute the time requirement is only dictionary searching that is  $T_{dic}$ , max determines the maximum time between these times.

Time requirements (if all attributes are present except the first attribute) in case of multiple attributes is  $T_{2orq} = \max \{ T_{dic}, (n-1) * T_{q/word} \}$  where n is the total number of attributes, here last (n-1) attributes are present but first attribute is absent. So for the first attribute the time requirement is only dictionary searching that is  $T_{dic}$ .

Time requirements (if first and last attributes are present but any attribute between first and last is absent) is  $T_{3orq} = 2 * T_{q/word} + (n-2)*T_{dic}$

In case of multiple attributes of AND operation where any attribute is absent, the total time requirements is

$$\begin{aligned} T_{orq} &= \max \{(n-1)*T_{q/word}, T_{dic}\} + \max \{T_{dic}, (n-1)*T_{q/word}\} + 2 * T_{q/word} + (n-2)*T_{dic} \\ &= 2 * \max \{(n-1)*T_{q/word}, T_{dic}\} + 2 * T_{q/word} + (n-2)*T_{dic}; \text{ where } (n-1)*T_{q/word} > T_{dic} \\ &= 2 * (n-1)*T_{q/word} + 2 * T_{q/word} + (n-2)*T_{dic} \\ &= 2 * n * T_{q/word} + (n-2)*T_{dic} \end{aligned}$$

The average case time requirements for OR operations is  $(T_{q(max)} + T_{q(min)})/2$

From 3.72 and 3.71 we see that the time requirements of AND operation is greater than that of OR operation.

### 3.8 Analysis of Memory Requirements for Various Methods

Let us consider, P be the total number of distinct element\_Paths in whole document, W be the total number of distinct words in whole document, D be the total number of documents .

**For storing integer data needs 2 byte.**

Memory needed for Word dictionary & Path dictionary is at least  $2 * (W + P)$  bytes.

Memory needed for Token\_Path\_word\_Matrix is  $(2 * D * (P + W))$  bytes

So, For Existing three dimensional indexing needs total memory:  $2 * P * W * D$  bytes.

For Two dimensional indexing needs total memory:  $(2 * D * (P + W) + W + P)$  bytes

For Bitmap Indexing needs total memory:  $((2 * D * (P + W) / 16) + (W + P))$  bytes.

Here we have considered 16 memory cells per block that means 32 bytes.

# Chapter 4

## Experimental Results and Discussions

### 4.1 Introduction

In this chapter we have explained memory requirements for three dimensional vs our proposed two dimensional bitmap indexing method. Time requirements of bitmap indexing and three dimensional indexing are described also. Words selectivity and words searching time are explained due to the variation of number of documents. This searching time increases due to increasing the document number. Path construction time of different types of file is explained in this chapter.

### 4.2 Experimental Setup

To implement our indexing method, we have used 2.6 GHz, Pentium-III processor system. Initially we have used 256 MB RAM in turboC compiler which supports 640K RAM including virtual memory, under Windows 98 operating system. Later on we have used BorlandC compiler which is 32 bit compiler having 4 GB RAM, under Windows XP operating system. We have used xml data repository from Internet given in [8] to implement and test our system. The dataset contains the three different kinds of information in three XML files namely the supplier information, the aerospace information and personnel information of an organization.

### 4.3 Memory Requirements for Various Indexing Methods

Using the dataset given in [8] we run our system and found the various numbers of words and paths that is given in Table-4.1. In Table 4.1 the first column represents the serial number of different datasets. In 2<sup>nd</sup> column there are three different datasets. Personal.xml dataset has total 36 distinct words and 15 distinct numbers of paths. Similarly Lineitem.xml dataset has 100000 words and 20 distinct numbers of paths, Nasa.xml dataset has 200000 words and 50 distinct numbers of paths. Using these datasets and according to memory requirements method given in 3.7, we found the table-4.2 and its corresponding graph is in Figure-4.1. In

Table-4.2 the first column represents the memory requirements for bitmap indexing of Personal.xml, Lineitem.xml and Nasa.xml respectively. The second column represents the memory requirements for two dimensional indexing of Personal.xml, Lineitem.xml and Nasa.xml respectively. Similarly the third column represents the memory requirements for three dimensional indexing of Personal.xml, Lineitem.xml and Nasa.xml respectively. In Table 4.2 the memory requirements for bitmap indexing is .51 MB and 195.31 for three dimensional indexing in case of Nasa.xml dataset. So the ratio of memory requirements of three dimensional vs bitmap indexing is about to 400:1. This is because in case of memory requirements of three dimensional indexing, all the paths are in one dimension, all the documents are in another dimension and the word information are stored in the third dimension. But in bitmap indexing all the paths and words are in column wise and all the documents are in row wise in a two dimensional matrix.

**Table 4.1: Dataset from xml repository**

Sl-no	Dataset	No_Of_distinct_words	No_Of_distinct_Paths
1	Personal.xml	36	15
2	Lineitem.xml	100000	20
3	Nasa.xml	200000	50

**Table 4.2: Memory needed for various indexing methods**

Dataset	Bit-Map Indexing (MB)	Two-Dimension (MB)	Three-Dimension(MB)
Personal.xml	0.000124	0.000498	0.0042
Lineitem.xml	0.256	1.17	19.53
Nasa.xml	0.51	4.29	195.31

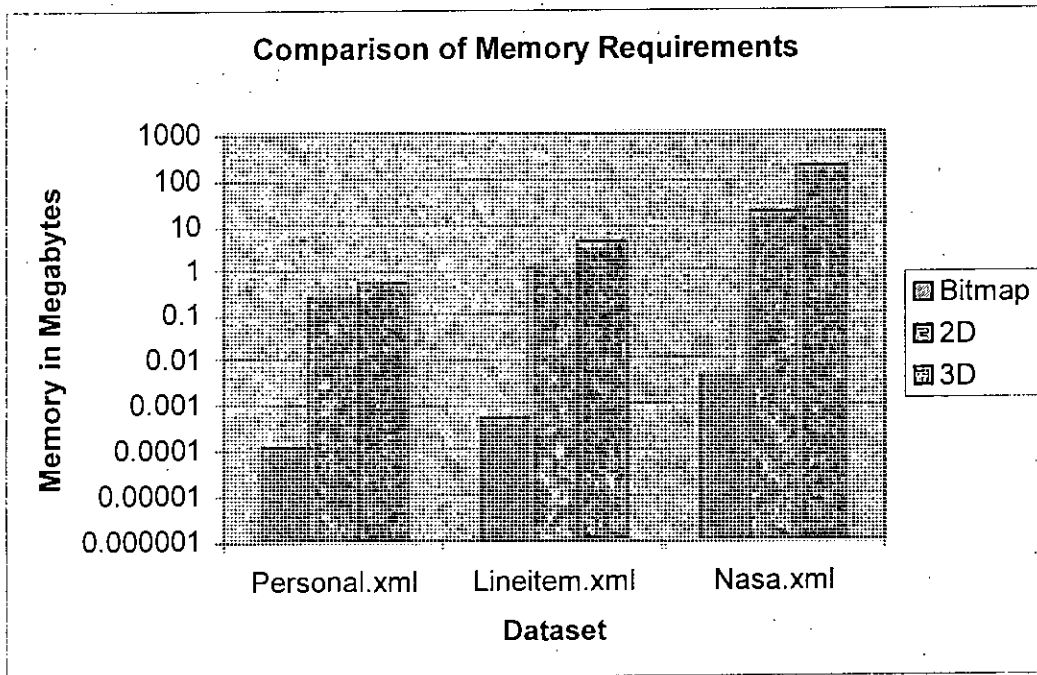


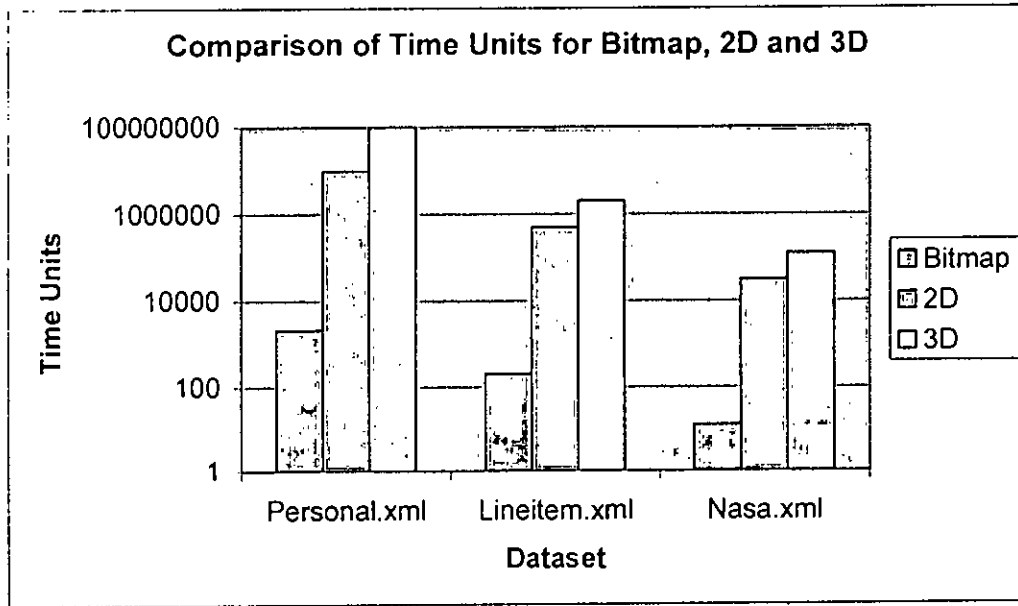
Figure 4.1: Comparison of Memory requirements for 3D, 2D and Bitmap indexing

#### 4.4 Time Requirements For Various Indexing Methods

Using the dataset in Table 4.1, according to time complexity described in 3.6, we get the time units of various indexing methods, given in table 4.3 and the corresponding graph is given in Figure 4.2. Here is an example to calculate the time units. Suppose for dataset Nasa.xml, the total numbers of distinct words are 200000, distinct element paths are 50 and the number of documents are 10. The time units required for three dimensional Bitmap indexing are  $O(200000 * 50 * 10) = 10^8$  units time, as the time complexity of the three dimensional bitmap indexing is  $O(\text{documents} * \text{Paths} * \text{Words})$ . Similarly time units are calculated for each indexing methods. Last column represents the ratio of the time units of 3D, 2D and bitmap indexing. In Nasa.xml, the ratio of time units of 3D: 2D: bitmap is 800:16:1. This is because time complexity of three dimension indexing is  $O(\text{documents} * \text{Paths} * \text{words})$  and for bitmap indexing is  $O(((W+P) / \text{no\_bits\_per\_block}) * d)$ . In our indexing method we have considered 256 bits per block that means 32 bytes.

**Table 4.3: Time unit needed for various indexing methods for various dataset**

Dataset	Average time Units for three Three-Dimensional Indexing	Average time Units for 2D Indexing	Average time Units for Bitmap Indexing	Ratio 3D:2D:bitmap
1	2160	204	12.75	170:16:1
2	10 <sup>7</sup>	500100	31256	320:16:1
3	10 <sup>8</sup>	2000500	125031	800:16:1



**Figure 4.2: Graph of Time Comparison for 3D, 2D and Bitmap indexing**

### 4.5 Word searching time

We have evaluated our indexing methods using dataset given in Table 4.2 to find the word searching time. The searching time is given in Table 4.4.

**Table 4.4: Word searching time**

Dataset	Words	Searching time in seconds
1	Supplier_Name	5.86
1	Supplier_Id	5.80
1	Category	5.66
2	Positional_item	5.25
2	International_item	5.62
2	Local_item	5.73
3	Photographic_zones	5.92
3	Astrographic_zone	5.84
3	Magnitude	5.67

## 4.6 Word Searching Time and their Selectivity in Documents

We have evaluated our indexing methods using Personal.xml (From xml data repository) given in [8], to find the word searching time, percentage of word's selectivity in documents given in Table 4.4. The relationship between words presence vs searching time is given in Figure 4.3. In Table 4.4, the first column represents the different number of distinct words. Second column represents their searching time in seconds. Third column represents there are total number of documents 28, used to run our system. Forth column represents the number of documents the word is present. Last column represents the selectivity of words in percentage. In Figure 4.3 the y axis represents the searching time of different words and the x axis represents the words selectivity in documents. The total graph represents the searching time requirements is linear. The word ICCIT is absent in all XML documents but searching time is .40 seconds. This searching time actually the time to search from word dictionary.

**Table 4.5: Word searching time and their selectivity in documents**

Words	Processors Searching Time in seconds	Total no of documents	Words selectivity in documents	Percentage of words selectivity
Dhaka	4.48		28	100 %
IEEE	3.75		26	92.85 %
ACMSIGMOD	3.56	28	25	89.28%
AI	3.25		23	82.14 %
TOHOKU	2.66		20	71.42 %
DATABASE	2.45		17	60.71%
WASHINGTON	1.94		10	35.71%
VLDB	1.75		7	25 %
Newyork	1.55		5	17.85%
MIT	1.01		2	7.14 %
BUET	.86		1	3.57%
ICCIT	.40		0	0 %

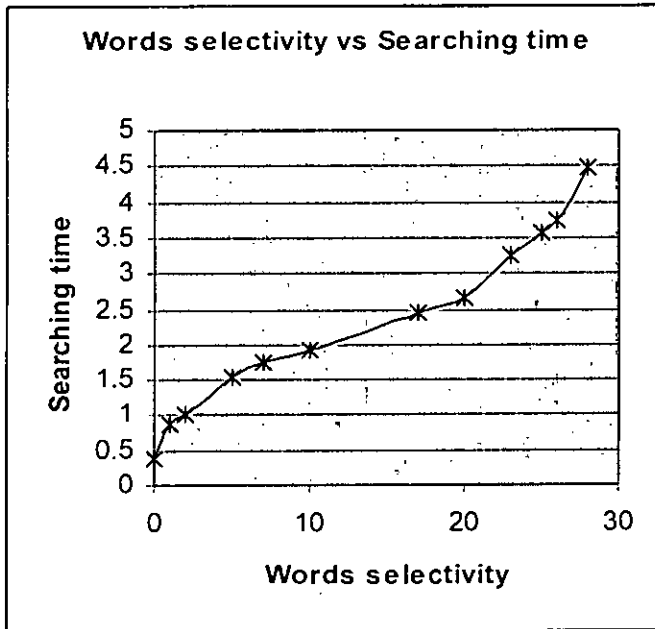


Figure 4.3: Searching time (sec) vs words selectivity Relationship

The word searching time, the percentage of word's selectivity in documents given in Table 4.5, and the relationship between words presence vs searching time is given in Figure 4.4 using dataset Personal1.xml. In Table 4.5, the first column represents the different number of distinct words. Second column represents their searching time in seconds. Third column represents there are total number of documents 56, used to run our system. Forth column represents the number of documents the word is present. Last column represents their percentage. In Figure 4.4 the y axis represents the searching time of different words and the x axis represents the words selectivity in documents. The total graph represents the searching time requirements is linear. The word MIST is absent in all XML documents but searching time is .82 seconds. This searching time actually the time to search from word dictionary.

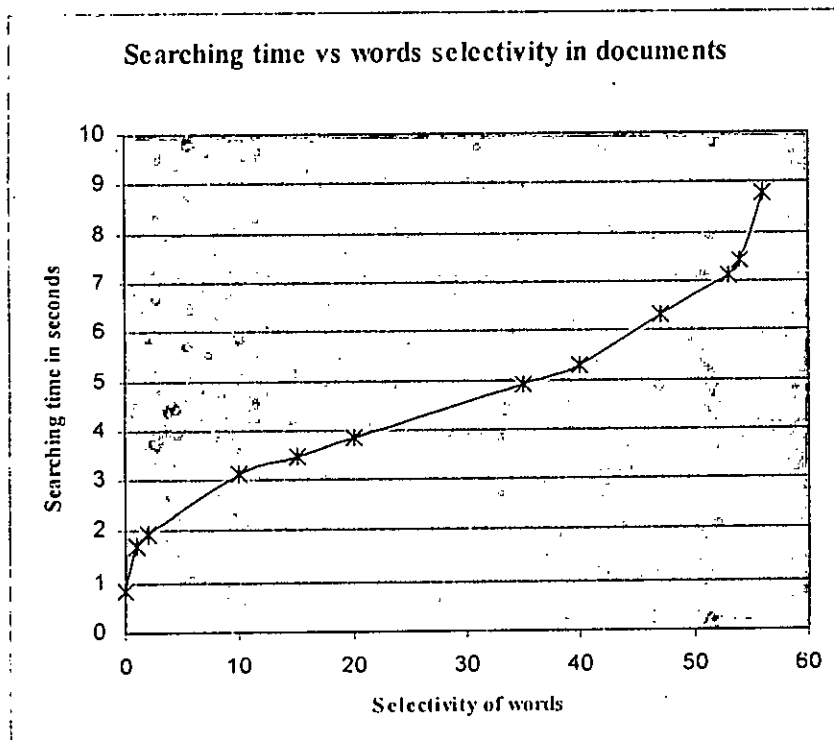
Comparing Table-4.4 and Table-4.5, the percentage of words selectivity is almost same but the total number of documents is double in Table-4.5 than that of Table-4.4. So the searching time is also almost double in Table-4.5 than that of Table-4.4. But considering the Figure-4.3 and Figure-4.4 it is clear that the searching time is linear for the words given in Table-4.4 and Table-4.5. The time complexity of bitmap indexing is :  $O(((W+P) / \text{no\_bits\_per\_block}) * D)$ . Where W be the total no of distinct words, P be the total no of distinct path, and D be the



number of documents. When D,W and P increases the searching time will be increased. So comparing Table-4.5 and Table-4.4, the searching time of Table-4.5 is greater than that of Table-4.4 as in Table-4.5 there are total number of 56 documents but in Table-4.4 only 28 documents.

**Table 4.6: Word searching time and their selectivity in large documents**

Words	Processors Searching Time in seconds	Total no of documents	Words selectivity in documents	Percentage of words selectivity
Paris	8.80		56	100 %
ICCIT	7.40		54	96.42 %
Datawarehouse	7.10	56	53	94.64 %
Multimedia	6.32		47	83.92 %
Saitama	5.30		40	71.42 %
RDMS	4.90		35	62.5%
Graphics	3.85		20	35.71%
IPSI	3.45		15	26.78 %
LORDS	3.10		10	17.85 %
Stamford	1.95		2	3.57 %
DUET	1.72		1	1.78%
MIST	.82		0	0 %



**Figure 4.4: Searching time vs words selectivity Relationship in large documents**

## 4.7 Searching Time for Multiple Attributes:

We have evaluated our indexing method with the personal.xml data for AND and OR operations given in Table-4.6 and Table-4.7

### 4.7.1 Query with AND operation:

In Table-4.6 first row represents query on the basis of different number of attributes. In second row, the searching time of those queries are presented. There are total number of documents is 30 to run these query. As for example in Table-4.6 the searching time of query with single attribute is 3.35 seconds but the searching time of query with two attributes is 4.08 seconds and the searching time of query with three attributes is 4.39 seconds. It is clear that the searching time of query with two attributes is more than the searching time of query with three attributes. Also the searching time of query with three attributes is more than the searching time of query with two attributes. This is because in multiple attributes there are more condition to satisfy the query.

**Table 4.7: Query in multiple attributes with AND operation**

	<b>Single attribute</b>	<b>Two attributes</b>	<b>Three attributes</b>
Query	Select Name From all documents Where Id=00414	Select Hall_Name, Room_no From all documents Where Id=00414 and Name= Mohiuddin	Select Dept, Year, Session From all documents Where Name=Alom and Hall_name=K.N.I and Id=00414
Time in seconds	3.35	4.08	4.39

### 4.7.2 Query with OR Operation:

In Table-4.7 first row represents query on the basis of different number of attributes. In second row, the searching time of those queries are presented. As for example in Table-4.7 the searching time of query with single attribute is 3.35 seconds but the searching time of query with two attributes is 3.75 seconds and the searching time of query with three attributes is 4.25 seconds. Analytically we see in 3.7.2 the searching time of AND operation is more

than the searching time with OR operation. From Table-4.7 and Table-4.6 we see that the experimental searching time of AND operation is more than that of OR operation.

**Table 4.8: Query in multiple attributes with OR operation**

	<b>Single attribute</b>	<b>Two attributes</b>	<b>Three attributes</b>
Query	Select Name From all documents Where Id=00415	Select Hall_name, Dept From all documents Where Room_no=2009 or Id = 00415	Select Dept, Year, Session From all documents Where Name=Alom or Hall_name=K.N.I or Id=00414
Time in seconds	2.92	3.15	3.84

#### 4.8 Path Construction Time

Our indexing method has been evaluated to find the path construction time using various dataset given in Table-4.8. First column of Table-4.8 represent various dataset, in second column there are total number of paths, in third column the file size of the dataset and the last column represents the path construction time in seconds. From Table-4.8, it is clear that when the file size increases path construction time also increases.

**Table 4.9: Path construction time for various dataset**

<b>Dataset</b>	<b>Total Path</b>	<b>File size (MB)</b>	<b>Path Construction time in seconds</b>
Contact_info.xml	15	1.16	4.21
Supplier.xml	26	2.2	6.25
Nasa.xml	50	5.6	11.18

## 4.9 Discussions

In case of memory requirements, from Table-4.1 and Figure-4.1, we see the memory requirement for three dimensional indexing is 195.31 MB where as two-dimensional indexing requires 4.29 MB and the bitmap indexing requires only 0.51 MB. The ratio of memory requirements of three dimensional vs bitmap indexing is almost on the factor of 400:1. This improvement of memory requirement is the result of using compressed representation in our method.

In time complexity analysis the ratio of average time unit among 3D, 2D and bitmap is 800:16:1 in case of dataset Nasa.xml. The word ICCIT and MIST given in Table-4.4 and Table-4.5 are absent in all XML documents but their searching times are 0.40 seconds and .82 seconds respectively. These searching times actually the time to search from word dictionary. In Figure-4.3 and Figure-4.4, the x axis represents the searching time of different words and the y axis represents the words selectivity in documents.

Comparing Table-4.4 and Table-4.5, the percentage of words selectivity is almost same but the total number of documents is double in Table-4.5 than that of Table-4.4. So the searching time is also about double in Table-4.5 than that of Table-4.4. Considering the Figure-4.3 and Figure-4.4, it is clear that the searching time is linear for the words given in Table-4.4 and Table-4.5. The time complexity of bitmap indexing is:  $O(((W+P) / \text{no\_bits\_per\_block}) * D)$ . Where W be the total no of distinct words, P be the total no of distinct path, and D be the number of documents. When D, W and P increase the searching time will be increased. So comparing Table-4.5 and Table-4.4, the searching time of Table-4.5 is greater than that of Table-4.4 as in Table-4.5 there are total number of 56 documents but in Table-4.4 only 28 documents.

Analytically we see in 3.7.2 the searching time of AND operation is more than the searching time with OR operation. From Table-4.7 and Table-4.6 we see that the experimental searching time of AND operation is more than that of OR operation. From Table-4.8, it is clear that when the file size increases path construction time also increases.

# Chapter 5

## Conclusions

### 5.1 Introduction

Semistructured databases unlike traditional databases do not have a fixed schema, largely evolving, self-describing and can model heterogeneity more naturally than either relational or object-oriented databases. Example of such self-describing data is XML. XML is a standard for representing and exchanging information on the Internet. Querying XML data requires an efficient indexing method. Conventional indexing methods such as Sparse and Dense indexing, Hashing, B+ trees are not satisfactory as the size of XML documents are very large and their types are different. So Bitmap indexing plays an important role for XML data.

Existing three-dimensional bitmap indexing of XML data requires large space. At the same time, querying of large XML documents database is difficult. To overcome these limitations, we have developed an indexing scheme of XML data using a two-dimensional Bitmap, providing the facility to store element-path, token and documents in a two dimensional matrix. This system contains two dictionaries; one is element-path dictionary having all the distinct element paths for all XML documents and another token dictionary containing token values for the distinct words. This indexing scheme creates a token-path-document matrix; showing the existence of XML data in specific document and in appropriate path. In this thesis we present how XML data, its path and document can be stored in a two dimensional bitmap and describe its performance over three dimension.

### 5.2 Contributions:

The main contribution of this thesis work is as follows:

- **Reduction of dimension in index structure:** In three-dimensional Bitmap indexing, three-dimensional matrix is required to store element-path, word and document number. Same information we can represent using a two dimensional structure.

- **Improvement of storage performance:** The ratio of memory requirements of three dimensional vs bitmap indexing is almost on the factor of 400:1.
- **Improvement of query performance:** Reduction of search time to query any XML data from XML document due to dimensionality reduction. In time complexity analysis the ratio of average time unit among 3D, 2D and bitmap is 800:16:1 in case of dataset Nasa.xml.
- **Querying the XML data in compressed format:** In this indexing method we can query the XML data in compressed format

In three-dimensional Bitmap indexing, three dimensional matrix is required to store element-path, word and document number. In two-dimensional indexing we require only a two-dimensional matrix, which can store element-path, existence of word and document number. This system creates a new column to get a new path from XML document. To get each distinct word within that path, a new column within that path boundary is created. There is a negative sign before path number to distinguish from token value. This system will set a 1 to the corresponding document number, when any word is present in that document number. If there is path repetition among documents no new column will be created. Only the token value of the word within that path will be stored. This process continues for all XML documents and a two dimensional matrix is created.

### 5.3 Future work

A well structured XML documents must have the following properties:

It contains one or more elements. It has just one elements (root element) that contains all the other elements. Its elements are properly nested inside each other (no element starts in one element and ends in another). The names used in its element start tags and end tags match exactly. The names of attributes do not appear more than once in the same element start tag. The values of its attributes are enclosed in either single or double quotes. The values of its attributes do not reference external entities, either directly or indirectly. Its entities are declared before they are used. In our two dimensional bitmap indexing method, to search any word or any query it is required well structured XML documents. There is further scope to

develop two dimensional bitmap indexing method for XML documents not in well structured form

During parsing of XML document we have considered that the document is syntactically correct. This is not always true in practice. The parsing method can be improved to handle the case where the documents are not syntactically correct.

## REFERENCES

- [1]. Yoon P.J, Raghavan V, Chakilam V. A Three-Dimensional Bitmap Indexing for XML Documents. In Journal of Intelligent Information Systems, Vol. 17, pages 241-254, November, 2001.
- [2]. Rizzlo F, Mendelzon A. Indexing XML data with ToXin. In research report, pages 31- 49, University of Toronto, Department of Computer science, CA, 2001.
- [3]. Philip J, Li Q, Moon B. XISS/R: XML indexing and storage System Using RDBMS. In proceedings of the 29<sup>th</sup> VLDB conference, pages 1073-1076, Berlin, Germany, 2003.
- [4]. Kratly M, Pokorny J, Snasel V. Indexing XML Data with UB-trees. In research report, pages 155-164, department of Computer Science, VSB-Technical University of Ostrava, Czech Republic, 2002.
- [5]. Quanzhong Li, Bongki M. Indexing and Querying XML data for Regular Path expressions. In proceedings of the 27<sup>th</sup> VLDB conference, pages 361-370, Roma, Italy, 2001.
- [6]. Raghav K, Bohannon P, Jeffrey F, Shenoy P. Updates for Structure Indexes of XML data. In proceedings of the 28<sup>th</sup> VLDB conference, pages 239-250, Hong Kong, China, 2002.
- [7]. Tataarinov I, Zachary G, Alon H, Daniel S. Weld. Updating XML Data. In ACM SIGMOD 2001, pages 413-424, May 21-24, Santa Barbara, California, USA.
- [8]. <http://www.cs.washington.edu/research/xmldatasets/www/repository.html>
- [9]. Abiteboul S, Buneman P, Suci D. Inverted index, In Proceedings of the International Conference on Database Theory, pages 377-395, 2000.
- [10]. Abiteboul S, Buneman P, Suci D. Data on the Web: from relations to semistructured data and XML. Morgan Kaufmann, 1999.
- [11]. Milo T, Suci D. Index structures for path expressions. In Proceedings of the International Conference on Database Theory, pages 277-295, 1999.
- [12]. Liefke H, Suci D. XMill: an efficient compressor for XML data. In Proceedings of the ACM SIGMOD International Conference on Management of Data, pages 153-164, 2000.



- [13]. Buneman P, Davidson B, Hillebrand G, Suciu D. Adding structure to unstructured data. In Proceedings of the International Conference on Database Theory, pages 336-350, 1997.
- [14]. Christophides V, Abiteboul S, Cluet S, Scholl M. From structured documents to novel query facilities. In Proceedings of the ACM SIGMOD International Conference on Management of Data, pages 313-324, 1994.
- [15]. Arocena G, Mendelzon A. WebOQL: Restructuring documents, databases and webs. Proceedings of the IEEE International Conference on Data Engineering, pages 24-33, 1998.
- [16]. Abiteboul S, Quass D, McHugh J, Widom J, Wiener J. The Lorel query language for semistructured data. International Journal on Digital Libraries, 1(1): 68-88, April 1997.
- [17]. Buneman P, Davidson S, Hillebrand G, Suciu D. A query language and optimization techniques for unstructured data. In Proceedings of the ACM SIGMOD International Conference on Management of Data, pages 505-516, 1996.
- [18]. Fernandez M, Florescu D, Levy A, Suciu D. A query language for a web-site management system. SIGMOD Record, 26(3): 4-11, 1997.
- [19]. Buneman P, Mary F, Suciu D. UnQL: a query language and algebra for semistructured data based on structural recursion. *VLDB Journal* 9(1): 76-110, 2000.
- [20]. Christophides V, Cluet S, Moerkotte G. Evaluating queries with generalized path expressions. In Proceedings of the ACM SIGMOD International Conference on Management of Data, 413-422, 1996.
- [21]. McHugh J, Widom J. Query optimization for semistructured data. Technical Report, Stanford University, 1997.
- [22]. McHugh J, Widom J. Query Optimization for XML. Proceedings of the International Conference on Very Large Databases, pages 315-326, 1999.
- [23]. Liefke H. Horizontal query optimization on ordered semistructured data. Informal Proceedings of the International Workshop on the Web and Databases, pages 61-66, 1999.

