Pakistan Journal of Applied Sciences 2 (3):416-421, 2002

Dynamic Indexing Algorithms For Structuring Linear Data

Imran Razzaq, Muhammad Sher
Department of Computer Science, International Islamic University, Islamabad, Pakistan

Abstract: Dynamic Indexing Algorithms provides a useful tool for structuring (in the form of tree) linear data structures (in the form of linear array) and also enclosing them with some Meta information. The main goal is to present an algorithm (based on some algebraic formulas) that can arrange the linear data in a structural format on basis of static conditions and dynamic conditions too.

Keywords: Upper bound, Lower Bound, Current Index, and Original Index, Content.

Introduction

In most of the file formats information is stored without any Meta information (- w3c) (- Beniot Marshal, 1999) and any logical structure (- w3c) (- Beniot Marshal, 1999) (- Sean Mc Grath, 1999). The objective of this paper is to provide algorithm that can retrieve information from linearly arranged or indexed data strings and enclose them with Meta information (- w3c) (- Beniot Marshal, 1999) and arrange it in a tree structure.

Initially all the information is arranged linearly in the form of linear array and indexed linearly. When converting this information to tree structured organized hierarchy, and then there is possibility of repetition of a single element. The repetition of a single element is based on two approaches.

- Static Repetition
- Dynamic Repetition

Static Repetition: In static repetition an element in the tree can repeat itself for predefined number of times. e.g. In a database of billing system, the number of bills are predefined.

Dynamic Repetition: In dynamic repetition an element in the tree can repeat itself for Number of iterations depending upon certain condition(s) e.g. in a database system of telecom bills the number of calls in each bill can vary and based on some dynamic conditions.

When converting a linear array of data to tree structured (intelligible format ref) Meta enclosed
format, an element (- w3c) can repeat itself either dynamically or statically or not. So we need onealgorithm to traverse a tree data structure with
dynamic and static element repetition behavior and another algorithm to re index the index of linear array of data, during each iteration of repeating element, so that in next coming iteration, next block of data is retrieved. These two algorithms collectively perform the structuring of linear data via dynamically indexing the data, so this algorithm is named as dynamic indexing algorithm.

Document Philosophy: Before we discuss the possibilities of structuring of linear lists or arrays of data, we must know about the electronic document philosophy and what really the term Document means in the digital world.

By and large, an "electronic document" consists of three distinct components

- Data Content
- Logical Structure
- Presentation

Data Content: The data content part of the electronic document represents the words themselves, it's the abstract data, no presentation, and no other information about how to present this data is included.

Logical Structure: The logical structure of the electronic document represents the document type and organization of its contents, and describes, what the information is (what this data means, i.e. meta information) how this information relates to the other pieces of information of the document and different constraints on the data it can posses.

Presentation: The presentation of the electronic document describes the way the information is presented to the reader, on a piece of paper, browser screen and via the voice synthesis. Also, which fonts or voice inflections are used for each piece of information and so on? Presentation contains all the rules that are applied to the abstract data to generate its rendition.

Now a word processor – especially WYSIWYG (what you see is what you get) word processors entwines content and presentation in a very tight embrace. Using such tools we create documents with a specific output in device in mind – typically a paper of particular width and height. As for structure – capturing what the information really is – the concept is hardly present at all.

The only structural information stored relates to the creation of the final paper output – details about page margins, font sizes, and so on. (- Sean Mc Grath, 1999)

Linear Data Formats: The traditional word processors and printing frame works create electronic document in format contains contents and presentation in very tight embrace e.g. Microsoft Word, Hyper Text Markup Language, Adobe Portable Document Format and so on. In these document formats the content data that is displayed in document is stored in a linear array of strings including presentation information (how that information will be displayed). Beside this linear data there are also some other contents like graphics, image, drawing etc.

Now the contents, these document contains do not have any Meta information (ret) and the logical coherence or relationship between the data. The main emphasis in such type of documents is on renderings of documents rather than processing the information they contain. For example the main usage of HTML documents is to render the information over web, for PDF documents is the rendering the document for printing frameworks etc.

The following are major disadvantages of linear document formats.

- The contents and presentation are tightly mixed up so, these documents are generated with keeping a specific output in mind.
- The information they contain is not intelligible. i.e. it

- The information cannot be queried.
- The information cannot be processed.

Structural Document Formats: The structural document formats contains their contents, the actual information, in hierarchical data model (- w3c) (- w3c) (- Beniot Marshal) (- Sean Mc Grath, 1999) in which the element consists of data and Meta information and elements also contain distributed and sub-elements. Thus the whole document is organized in the form of tree of elements.

Such types of documents fulfill all the characteristics of electronic document philosophy. The examples of such documents are XML, SGML... These types of documents separate the contents with presentation and also preserve the structural information of document. There are primarily two applications of such type of structural documents like XML/SGML etc.

•Document Application (applications manipulate information that is primarily intended for human consumption.)

•Data Application (manipulate information that is primarily intended for software consumption.)

Document Application: The first application of structural document would be document publishing it has following features

- Independent of the delivery medium
- Edit and maintain documents in structural documents and automatically publish them
- The ability to target multiple media
- Maintain a common version of the documentation in a media-independent format, such as XML

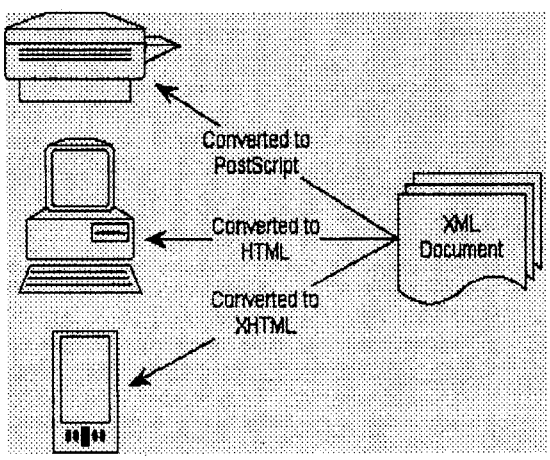
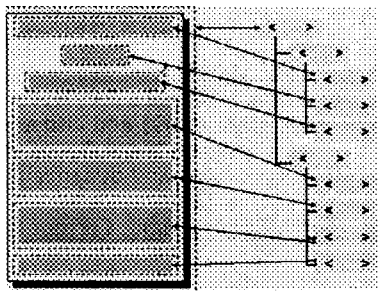


Fig. 1: Rendering XML Data

Data Applications: One of the original goals of structural document e.g. XML was to give document management access to the software tools that had been used to manage data, such as databases. With structural document, the loop has come to a full circle because structural document brings a publishing kind of distribution to data. This leads to the concept of "the application as the document" where, ultimately, there is no difference between documents and applications.

Indeed, if the structure of a document can be expressed in e.g. XML, as illustrated in Fig. 2.

Upper Bound



Lower Bound

Fig. 2: XML Document Application

So can the structure of a database, as illustrated in Fig. 3.

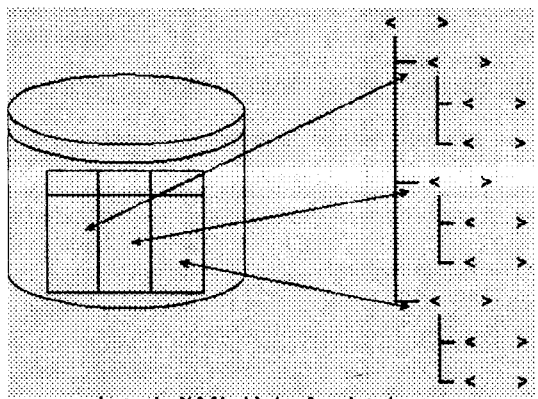


Fig. 3: XML Data Application

Transforming Information From Linear Data Structures: When transforming the information from linear data formats to structural data formats (in other words constructing logical structural tree (- w3c) (- w3c) (- Beniot Marshal) (- Sean Mc Grath, 1999) of document) we need first of all Meta Information and the it's children and some for it's children. For example the following document hold Students record and we want to convert it to a XML document.

Linear Format	Structural Format
0 – Name	<students>
1 - John	<student>
2 - Class	<name> John </name>
3 - M.Sc	<class> M.Sc </class>
4 – Reg#	< Reg#>218< /Reg#>
5 – 218	<Subjects>
6 - 1	<Subject>
7 – Math	<Title> Math </Title>
8 - 2.5	<GPA> 2.5 </GPA>
9 – P	<Status> P </Status>
10 – 2	</Subject>
11 – Stat	<Subject>
12 - 0.5	<Title> Stat </Title>
13 - F	<GPA> 0.5 </GPA>
	<Status> P </Status>
	</Subject>
	</Subjects>
	</Student>

As we noted in the above example that if we want to

construct the tree given at right side from the linear data from left side.
Then two tasks will be performed

- The tree traversal with dynamic and static repetition of a node or element (as subject repeats and students repeats)
- The content retrieval mechanism that can retrieve contents for specific node or element.

Now the user will specify the single instance of each element in hierarchy and will associate some properties. If an element is to be repeating then user will specify its upper bound and lower bound. Upper bound specify the starting index and lower bound specify the ending index of contents interval in which the elements can contain contents. For next iteration that interval will move forward. e.g. in the above example we can see that the upper bound of subject element must be 6 and lower bound must be 9. Now the subject can contain any content in between these bounds and for next subject instance we will move this bound to 10 to 13 so the next instance of subject can contain next piece of subject data.
So in dynamic indexing algorithm there are following three building blocks.

- Repetitive Tree Traversal Algorithm
- Dynamic Indexer Algorithm
- Index Retriever Algorithm

Repetitive Tree Traversal Algorithm: The repetitive tree traversal algorithm is an algorithm that can traverse the tree with capabilities of repeating a particular node for a number of iterations either specified statically or dynamically. When processing each node it will invoke "Index Retrieval Function" based on some algorithm described later to retrieve the contents that it will contain. And for repetition of a node, during each iteration, it will invoke "Dynamic Indexer Algorithm" to re-indexing the contents, so that next set of contents is retrieved. The Algorithm in pseudo code is described as follows

Repetitive Tree Traversal Algorithm

Note: The following is the pseudo code representing the complete algorithm of repetitive tree traversal, which is based on ordinary tree traversal algorithm and on stack rather than recursion.
ParentNode represents the Parent of current node and CurrNode represents the current node that's being processed. The ChildNode represents the child node of the current node. The RootNode represents the root node of the tree of nodes to be processed.
The FirstChildIndex attribute of tree node represents the first child of the node and NextSiblingIndex represents the index of next child of its parent node. ChildrenList provide all the facilities of a list and maintains all the children of a node.

```
//start from root
Step - 1        ParentNode ←_ NULL;
Step - 2        ChildNode  ←_ NULL;
Step - 3        CurrNode   ←_ RootNode;
Step - 4        Do
      Begin
                // Process the Node
Step - 5                ProcessNode(Parent, ChildNode);

                //get all the children of the CurrentNode
                //====================================

                //Get the first child
Step - 6                If CurrNode → FirstChildIndex = -1 then
                        return TRUE
                EndIf
Step - 7                ChildNode ← GetNode (CurrNode →_ FirstChildIndex)
Step - 8                ChildrenList → Empty ( )

Step - 9                Repeat while ChildNode != NULL
                Begin
                        ChildrenList → Add(ChildNode)
                        If ChildNode → NextSiblingIndex = -1 then
                                ChildNode ← NULL
                        Else
                                ChildNode ← GetNode (CurrNode → NextSiblingIndex)
                        EndIf
                End While
                //====================================
```

```
                //push all the values from the ChildrenList to stack in reverse order
Step – 10       Repeat Step – 4 for I = ChildrenList_Size → 1, ----, 2, 1, 0

                        Stack →Push (ChildrenList →_GetNode (I)
                }

                //pop one by one

Step – 11       Do Repeat
                Begin

                        ChildNode ←_ stack →Pop( )
Step – 12               If ChildNode = NULL then
                        Begin
                                // stack is empty
                                CurrNode ← NULL
                                ChildNode ← NULL
                                break
                        End If

Step – 13               FLAG = DoRepeat(ChildNode)
Step – 14               If FLAG = TRUE || ChildNode_←IsRepeating == FALSE)
                        Begin
                                If ChildNode ← IsRepeating != FALSE)
                                        Stack_Push (ChildNode)
                                End If

Step – 15                       If ChildNode → FirstChildIndex != -1 then
                                Begin
                                        // it contains the further child elements so process them first
                                        ParentNode ← CurrNode
                                        CurrNode ← ChildNode
                                        break
                                End
                                Else
                                Begin
                                        // it dont contains the child element so insert it in the tree
                                        ProcessNode(ChildNode, Node)

                                End If
Step – 16                       If ChildNode = NULL then
                                        CurrNode ←NULL

                        End If

                        End While ChildNode != NULL

                End While CurrNode != NULL
```

In the above algorithm the ProcessNode function is called to process a particular node of the tree. In the ProcessNode the Index Retriever Formula is used to retrieve the Content data for the node from the linear data list. This formula is described in following section.
The DoRepeat function in the tree traversal algorithm basically indexes the linear data list so that for a particular iteration of a repeating node the appropriate data content must be retrieved. This Algorithm is described in following section.
Dynamic Indexer Algorithm: The dynamic indexer algorithm indexes the linear array of contents so that the exact contents must be retrieved by the content retrieval during the processing of the Dynamic Tree Traversal Algorithm (described above).
The Repetitive Tree Traversal Algorithm invokes the DoRepeat function to check for the repetition of the particular node, in DoRepeat function the Dynamic Indexer Algorithm is executed which first of re-indexes the linear array of data contents so that the for the current iteration of node, the respective block of data must be considered and all the retrieval of data must be done from that block.

Dynamic Indexer Algorithm

The Dynamic Indexer Algorithm checks for the repetition of a node and also performs the respective re indexing of the linear data array.

```
If CurrNode → IsDynamic = TRUE then
Begin
        // repeatition is static
        If CurrNode →CurrIteration   <   CurrNode →NoOfIterations   then
        Begin
                If CurrNode →CurrIteration == 1 then
                Begin
                        // the first iteration
                        Content ←      GetContentByIndex (CurrNode →UpperBoundry)
                        CurrNode →StartIndex ←      Content →OriginalIndex
                        CurrNode_CurrIteration++
                        return TRUE
                End
                Else
                Begin
                        Repeat For I = UpperBoudry to LowerBoundry
                                CurrContent ← _  GetContent(I)
                                If CurrNode→_CurrIteration == 2 then
                                CurrContent→Stack→Push(CurContent_Index)
                                End If
                                CurrContent→_Index ← -1
                        End Repeat
                        NextStart ←  GetOriginalIndex
                                        (GetContent(LowerBound)→_OriginalIndex+1)
                        Increment ← LowerBound - UpperBound + 1
                        NewIndex ← iUpperBound;
                        Repart for I = nextstart to NextStart + Increment
                                CurrContent    ←     GetContent (I);
                                CurrContent→_Stack  →_Push(CurrContent_MIndex);
                                CurrContent→_MIndex      ←_  NewIndex;
                                NewIndex++;
                        End Repeat
                        CurrNode→_CurrIteration++;
                        return TRUE;
                End  If
        End If
```

```
Else
        Begin
                // it's the last iteration
                CurrNode→StopIndex  ←  GetContentByOriginalIndex

                                        (GetContent(CurrNode_LowerBoundry)
                //reinitialize the repeating
                CurrNode→CurrIteration ←  1
                ResumeIndex (CurrNode→_StartIndex, CurrNode→_StopIndex)
                return FALSE;
        End
EndIf
Else
        Note: Same work in Dynamic Repition but condition will not by no. of iteration but it can by any condition
End
```

In the above algorithm the Index Retriever Algorithm retrieves the Content on a particular index, which is described as follows.

Index Retriever Algorithm: The content in a linear array maintains two types of indexes, first one is the Original Index and second one is Modifiable Index or Current Index. Now the Index Retriever Algorithm is used for retrieving the contents based on both type of indexes, current index and permanent index. Following is the algorithm.

Index Retriever Algorithm

The DocArray represents the list of contents that is a linear array. It provides all facilities of a list.

```
Repeat For I = 0,1,2, .... , DocArray→Size
Begin
        Content ← DocArray→_GetAt(I)
        If Content→_Index = ReqIndex then
                return Content
End For
Retrun NULL
```

Same Algorithm is used to retrieve the content with respect to original index .

The above three algorithms, Dynamic Tree Traversal Algorithm, Dynamic Indexer Algorithm and Index Retriever Algorithm collectively work to perform the structuring of linear data. In this whole solution, the Dynamic
Tree traversal Algorithm traverses the structural tree and invokes the Dynamic Indexer Algorithm to update the indexes of linear list of contents and uses the Index Retriever Algorithm to retrieve the content for a particular node.

Conclusions

These algorithms provide a successful and reliable mechanism for structuring of data. This algorithm is used in PDF to XML conversion solutions, in Database to XML conversion solutions etc. But still certain enhancements are expected in the algorithms for the improvement of their efficiency.

Acknowledgements

The authors wish to thank Almighty Allah who enabled them to do such type of research work. They will acknowledge the technical help of Mr. Muhammad Tauheed (Senior Manager, Elixir Technologies Pakistan) and Prof. Dr. Khalid Rasheed (Head, DCS, IIUI).

References

- W3C (World Wide Web Consortium), "standard for structured document formats e.g. XML", www.w3c.org
- W3C (World Wide Web Consortium), "Standard for traversing the tree-structured document formats e.g. XML", www.w3c.org
- Beniot Marshal, "XML By Examples", 1999.
- Sean McGrath, "XML By Examples", 1999.
- Adobe Systems Incorporated, "Portable Document Format Reference Manual 1.3", March 1999.