Universidade Federal de Campina Grande
Centro de Ciências e Tecnologia
Curso de Pós-graduação em Informática

Indexando XML

Banco de Dados e Internet
Prof. Cláudio Baptista
Rômulo Nunes

Campina Grande mai/2004

Índice

Índice de Figuras

Resumo

Hoje em dia existem vários SGBDs (Sistemas de Gerenciamento de Banco de Dados) no mercado que dão suporte a documentos XML. Alguns são nativos XML (Tamino) e outros (Oracle, Microsoft SQL Server, IBM DB2) prometem trabalhar com vários tipos de base de dados, tendo XML como uma delas. Quando se trabalha com XML é necessário lidar com arquivos, e mais: saber como extrair de forma eficiente os dados contidos no arquivo. O que será abordado neste relatório são as soluções existentes para realização de uma consulta em um documento XML, de forma rápida e eficiente.

Palavras-chave: WWW, indexação, XML, mecanismos de busca, algoritmos de busca.

Abstract

Nowadays there are some DBMSs (Data Base Management System) in the market that supports XML documents. Some of these DBMSs are XML native (Tamino) and others (Oracle, Microsoft SQL Server, IBM DB2) promise to work with many data types, including XML. When one works with XML, it is necessary to deal with files, and more: We have to know how to extract the data contained in these files in an efficient way. In this report, it will be covered the actual solutions to process queries in an XML file, in a fast and efficient way.

Key-words: WWW, indexing, XML, search engines, search algorithms.

1- Introdução

Documentos XML têm ganhado espaço no mercado a cada dia. Hoje em dia existem vários SGBDs no mercado que dão suporte a XML. Mas quando se trabalha com XML é necessário lidar com arquivos e mapea-los de forma coerente em árvore estrutural de dados XML. Podemos realizar buscas em documentos por diversas formas, onde a mais simples seria a leitura de um documento do inicio ao fim. Mas estamos falando em dados para tráfego na internet e o tempo da busca por uma informação é algo muito importante. A indexação dos documentos é a saída mais visada para resolver este problema. Indexar não é só o problema, temos que nos preocupar com a recuperação dos índices de forma eficiente e veloz.

2 - Analogia dos documentos XML com uma lista.

A análise estruturada nos ensina alguns métodos para se manter uma lista ordenada e conseqüentemente mais fácil de se realizar buscas. É certo afirmar que o tempo empregado para essa organização prévia dos dados tem seu custo. Como queremos priorizar a coleta de informações em um tempo mínimo, devemos investir na organização dos dados nos documentos.

```
<?xml version="1.0" ?>
<agenda>
        <contato Nome=”Ana Paula da Silva”>
                <Endereço>
                        Rua Zacarias de Azevedo, 323 Prado.
                </Endereço>
                <Telefone>
                        323-3310
                </Telefone>
                <Telefone>
                        9991-3817
                </Telefone>
        </contato>
        <contato Nome=”Cláudia Oliveira”>
                <Endereço>
                        Av. Duque de Caxias, 454 Centro.
                </Endereço>
                <Telefone>
                        355-3715
                </Telefone>
        </contato>

                ...

        <contato Nome=”Zelda Cavalcante”>
                <Endereço>
                        Av. Monte Castelo, 1364 Levada.
                </Endereço>
                <Telefone>
                        223-2233
                </Telefone>
        </contato>
</agenda>
```

(figura 1 – Código XML simples de uma agenda)

Contato 1	Contato 2	Contato 3	...	Contato n
Ana Paula	Cláudia	Débora		Zelda

(figura 2 – Lista de elementos “Contatos”)

Quando observamos um determinado nível da árvore do documento (as tags), podemos compara-la a uma lista *(figura 1 e figura 2).* Cada elemento da lista corresponde a uma sub árvore na estrutura do documento XML. Um apontador (índice ou ponteiro) para o documento original também é guardado na lista, isso quando não é carregada toda a sub árvore para a memória.

2.1 - Para a ordenação da lista

Existem vários algoritmos conhecidos da estrutura de dados, dentre eles o Quick Sort (método de Hoare) é um dos mais eficientes. Ele trabalha de forma recursiva subdividindo a lista em listas menores. Observe que o "segredo do sucesso" deste algoritmo esta na escolha de um bom pivô a cada chamada recursiva. Depois de ordenado, um elemento no documento pode ser facilmente encontrado. Por exemplo, bastaria aplicar uma busca binária.

2.1.1 - Algoritmo de Ordenação

a) Caso básico: Se o número de elementos a ordenar for 0 ou 1 então terminar
b) Seleccionar o elemento Pivô "P": Escolhendo um elemento qualquer entre os elementos a ordenar
c) Processo de partição: Dividir os elementos em 2 subconjuntos disjuntivos na qual:

m= (x Î vector - (P) | x £ P) //elementos inferiores ao pivô

M= (x Î vector - (P) | x > P) //elementos superiores ao pivô

d) Método recursivo: Ordenar os subconjuntos esquerdo e direito, usando o mesmo método recursivamente.

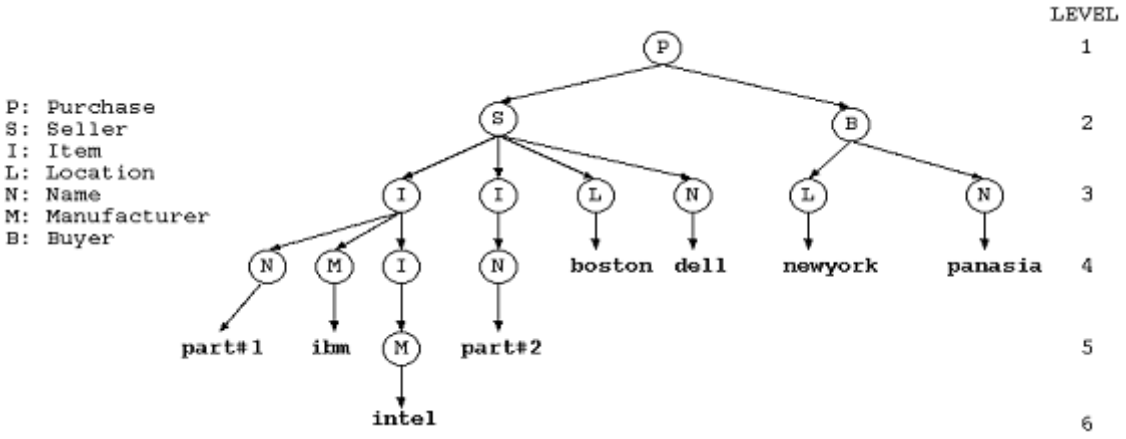
2.1.2.-.Algoritmo de Partição

a) Escolha do pivô: Considere p=a[lim_inferior] como o elemento pivô. Usa-se o primeiro elemento para facilitar a implementação.
b) Dois ponteiros *alto* e *baixo* são inicializados como os limites superior e inferior do array/vector a ordenar. Em qualquer ponto de execução, todo o elemento acima de *alto* é maior do que *x* e todo o elemento abaixo de *baixo* é menor do que *x*.
c) Os dois ponteiros alto e baixo são movidos um em direção ao outro da seguinte forma:
    - Incrementa baixo em uma posição enquanto que *a[baixo] £ p*.
    - Decremente alto em uma posição enquanto que *a[alto] > p*.
    - Se *alto > baixo* , troque *a[baixo]* por *a[alto].*

    O processo é repetido até que a condição descrita mais acima falhe (quando *alto <= baixo*). Neste ponto *a[alto]*será trocado por *a[lim_Inferior]*,cuja posição final era procurada, e alto é retornado em i.

3 - Analogia dos documentos XML com uma árvore

De uma forma mais geral, considerando todos os elementos do documento XML, podemos compara-lo a uma árvore. A hierarquia da árvore é comparada à hierarquia das tags do documento (de fora para dentro). Veja na *figura 3* um exemplo de um documento XML mapeado em uma árvore.



(figura 3 – Árvore que representa um documento XML)

De alguma forma a árvore precisa ser também descrita como um conjunto de nós. Ou seja, queremos construir uma forma de representar a árvore como um encadeamento de nós. Percorrendo a árvore em pré-ordem temos uma forma simples de representar. Considerando as variáveis de v1 à v8 a representação das folhas da árvore, a árvore da *figura 3* ficaria:

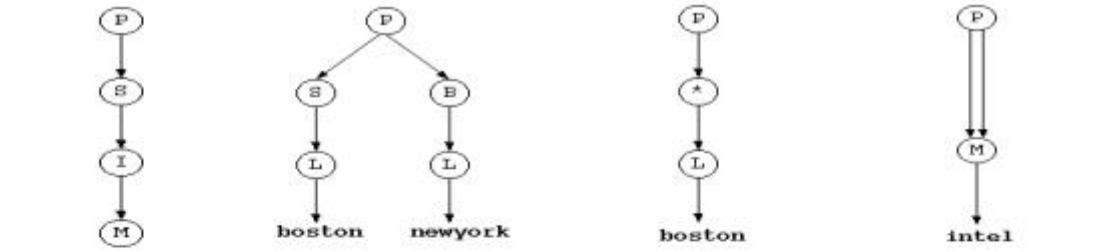
$$PSINv_1Mv_2IMv_3INv_4Lv_5Nv_6BLv_7Nv_8$$

Uma notação mais aprimorada é a que vamos usar para representar cada nó da árvore. Também percorremos a árvore em pré-ordem e usamos tuplas (símbolo_atual , prefixo) para representar os nós. Esse método de representação chama-se Structure Encoded Sequence ou, em português, Seqüência Estruturada Codificada. A árvore agora fica representada assim:

D = (P,e),(S,P),(I,PS),(N,PSI),(v1,PSIN),(M,PSI),(v2,PSIM),(I,PSI),
(M,PSII),(v3,PSIIM),(I,PS),(N,PSI),(v4,PSIN),(L,PS),(v5,PSL),
(N,PS),(v6,PSN),(B,P),(L,PB),(v7,PBL),(N,PB),(v8,PBN)

As consultas para serem utilizadas têm que ser convertidas para uma seqüência estruturada codificada incluindo consultas ramificadas. Podemos

também utilizar caracteres coringas ‘*’ ou ‘//’ para a busca. Onde ‘*’ substitui um nó na estrutura hierárquica da árvore e o “//” substitui vários nós.



Procurar todos os fabricantes que fornecem itens

Procurar compras com Vendedor Boston e comprador NY

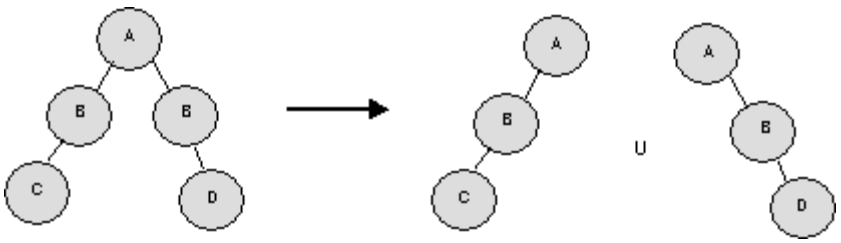
Procurar compras com Boston como comprador ou vendedor

Procurar compras que contêm produtos Intel

	Path Expression	Structure-Encoded Sequence
Q_1:	$/Purchase/Seller/Item/Manufacturer$	$(P,\epsilon)(S,P)(I,PS)(M,PSI)$
Q_2:	$/Purchase/[Seller[Loc=v_5]]/Buyer[Loc=v_7]$	$(P,\epsilon)(S,P)(L,PS)(v_5,PSL)(B,P)(L,PB)(v_7,PBL)$
Q_3:	$/Purchase/*/[Loc=v_5]$	$(P,\epsilon)(L,P*)(v_5,P*L)$
Q_4:	$/Purchase//[Manufacturer=v_3]$	$(P,\epsilon)(M,P//)(v_3,P//M)$

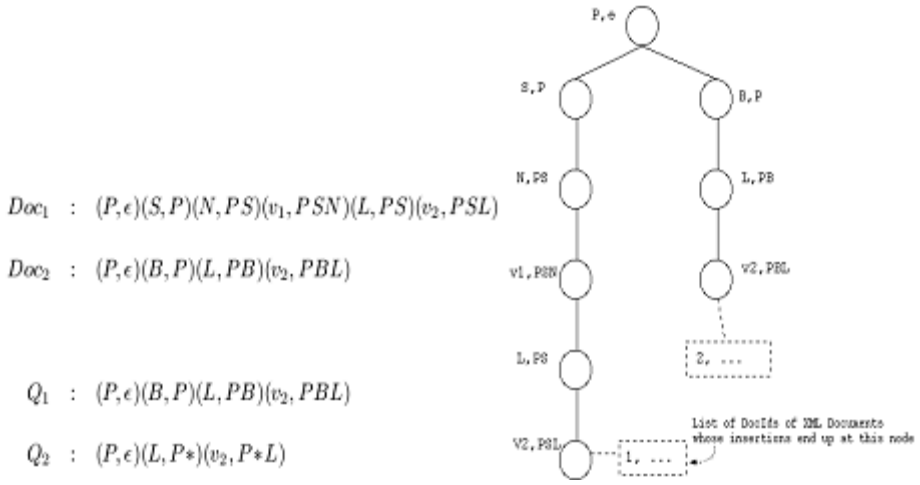
(figura 4 – Exemplo de consultas)

Um problema em se fazer consultas assim é que a linguagem ainda é pobre. Não podemos fazer buscas complexas a não ser que o problema seja dividido em outros e depois usemos o conjunto união ou diferença para uma resposta final. Por exemplo, uma consulta que deve ser solucionada usando a decomposição é a /A[B/C]/B/D, que na notação que usamos para consulta fica: (A,e), (B,A), (C,AB), (B,A), (D,AB). O problema é que o nó pai (A) tem filhos parecidos (B).



(figura 5 – Ilustração de decomposição de consulta)

Para resolver problemas desta natureza utilizaremos uma árvore trie para guardar a seqüência de sufixos da árvore. Observe na *figura 6* dois documentos representados numa árvore trie. Cada nó é um elemento da notação de seqüência estruturada codificada. No final de cada documento é guardado o índice para o acesso físico dele. Lembre-se que estamos trabalhando na memória principal.



(figura 6 – Representação de documentos na árvore trie)

3.1 - Algoritmo Simples de Busca

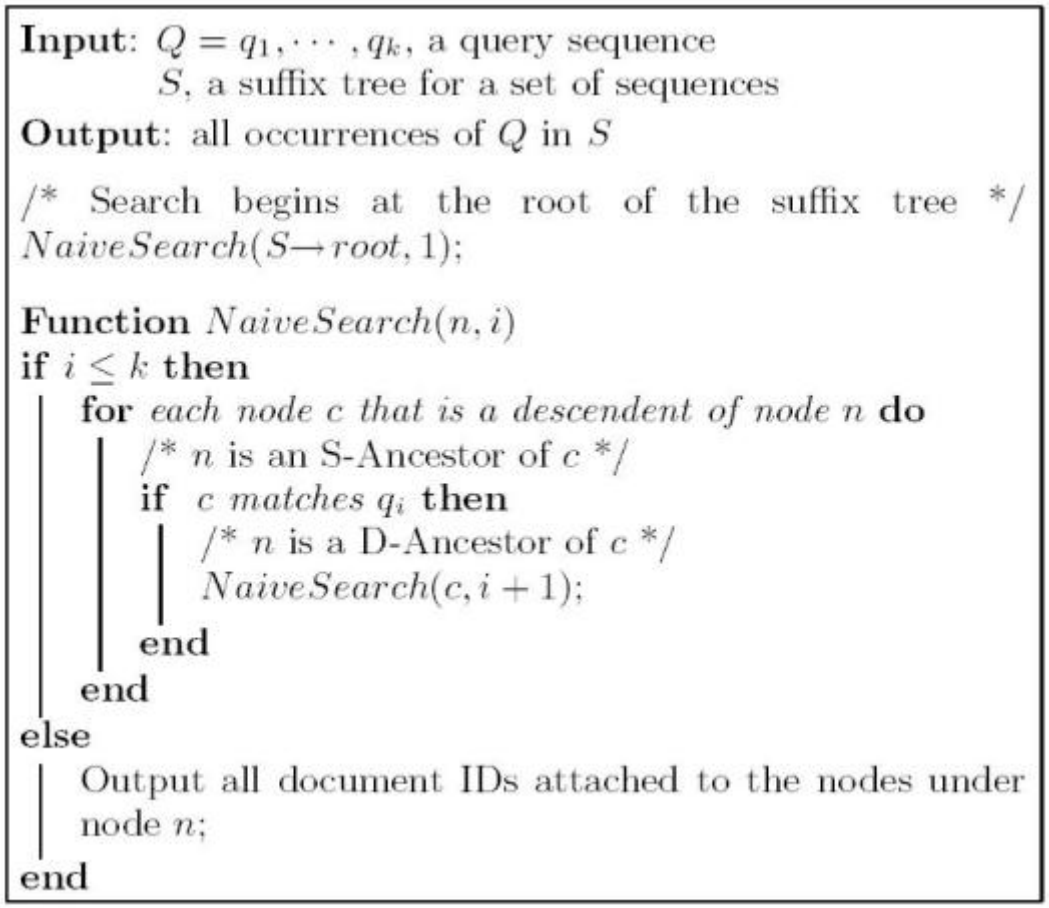
Um dos algoritmos de busca para varrer uma árvore representada por uma seqüência estruturada é o exibido na *figura 7*. O algoritmo é bem simples onde cada nó da sub árvore é comparado com a consulta, que também é uma sub árvore.

O problema com a solução do algoritmo da *figura 7* é que ele precisa percorrer todos os nós da árvore para ter uma resposta. Existe outro problema que torna a solução inviável para grandes bases de dados e para sistemas de gerenciamento de banco de dados. É que toda árvore precisa ficar carregada na memória principal. Isso torna a solução pouco adotada pelos analistas.

3.2 - RIST – Relationships Indexed Suffix Tree

O método RIST veio para suprir os problemas anteriores. Mas para implementa-lo precisamos marcar a árvore trie mais uma vez. Percorrendo a árvore em pré-ordem marcamos cada nó com seu tamanho e ordem de classificação. Cada nó recebe uma tupla (pré(no) , size(nó)), onde a função pré() retorna a classificação em pré-ordem do nó, e a função size() retorna o

seu número de filhos. A idéia é alcançar o primeiro nó de ocorrência X e em seguida ser lançado para uma outra possível ocorrência de X em um nó Y que é “filho de um nó irmão do pai de X” ou seja, que teoricamente está no mesmo nível



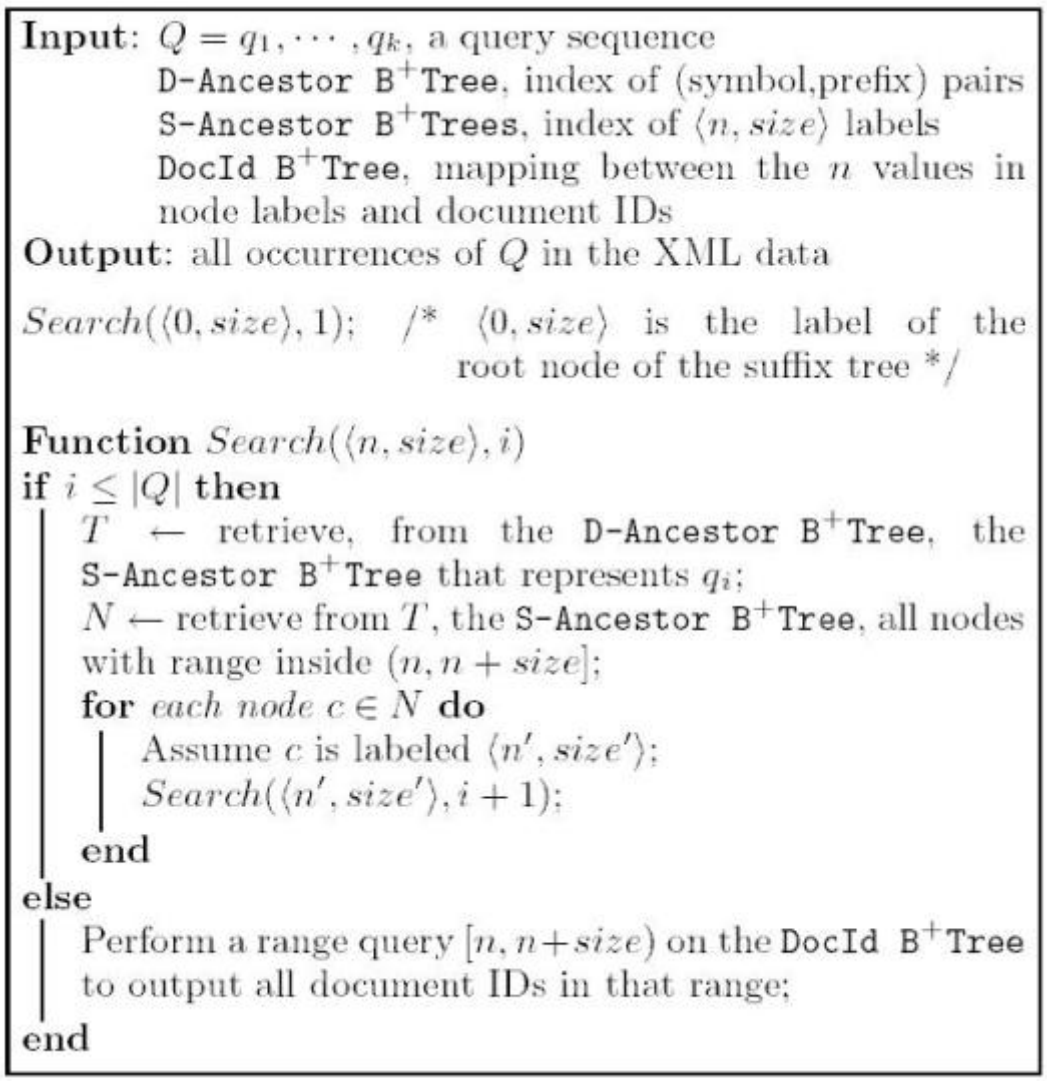
```
Input: Q = q_1, ..., q_k, a query sequence
       S, a suffix tree for a set of sequences
Output: all occurrences of Q in S

/* Search begins at the root of the suffix tree */
NaiveSearch(S→root, 1);

Function NaiveSearch(n, i)
if i ≤ k then
    for each node c that is a descendent of node n do
        /* n is an S-Ancestor of c */
        if c matches q_i then
            /* n is a D-Ancestor of c */
            NaiveSearch(c, i + 1);
        end
    end
else
    Output all document IDs attached to the nodes under
    node n;
end
```

(figura 7 – Algoritmo de uma busca simples)

O problema é que o método RIST utiliza um esquema estático para guardar as tuplas (pré(nó) , size(nó)), isso impede que novos nós seja inseridos dinamicamente, pois isso acarretaria em atualizar todos os nós da árvore. RIST precisa de uma árvore de sufixos totalmente carregada na memória principal, isso o torna não suportado pelos SGBDs. Em 2002 Edith Cohen, Haim Kaplan e Tova Milo apresentaram uma solução. Como conseqüência surgiu o método de busca VISIT, que trabalha de forma semelhante ao RIST, com a diferença que os índices de busca são alocados dinamicamente. Com isso é possível adicionar novos nós a árvore sem problemas.



(figura 8 – Algoritmo do método RIST)

4 - Outras pesquisas sobre indexação

Muitas pesquisas no mundo sobre indexação de XML estão acontecendo. O VIST é uma das mais recentes divulgadas. Um outro grupo de pesquisa, na Universidade de Wisconsin-Madison, propõe uma forma de consulta mista em documentos XML. Eles justificam essa proposta pelo fato de que consultas mais simples em sistemas XML nativos podem ter resultados melhores com combinação da navegação baseada em árvores e estrutural. Na Universidade de St. Petersburg na Rússia existem pesquisas com “Indexing XML to Support Path Expressions”, na Universidade de Singapura com “XR-Tree: Indexing XML Data for Efficient Structural Joins”, “XSEarch: A Semantic Search Engine for XML” em Jerusalém - Israel, e outros; Nas universidades do Brasil: Pesquisas de indexação XML para fins específicos como bibliotecas digitais (Padrões em Bibliotecas Digitais – Universidade

Católica do Rio Grande do Sul), sistemas de busca (Projeto indexa – Universidade de Minas Gerais), e outros.

5 - Conclusão

Pesquisas na área de indexação fortalecem a utilização do XML em SGBDs, sistemas multi-plataforma e aplicações web. Além de incentivar a sua utilização como principal ferramenta de armazenamento de dados para na internet. A evolução nos métodos de consulta não parou aqui. As pesquisas convergem em direção a uma forma única, global e eficiente de busca em um documento XML. É cedo para determinar o método ou idéia ideal.

Bibliografia

[1] - B. Magill C.Winstead CSE582 – Indexing XML - http://www.cse.ogi.edu/class/cse582/Lectures/Lecture12/XML%20Index%202.pdf (acessado em 29/04/2004)

[2] - Fernanda Baião – (baião@cos.ufrj.br) Publicação de Dados de SGBDs em XML - Agosto de 2003 http://www.cos.ufrj.br/~baiao/XML-LabBDI20032/Aula8_PublicacaoDadosXML.pdf (acessado em 30/04/2004)

[3] - Hamid R. Shahbazkia - Quick Sort - http://w3.ualg.pt/~hshah/ped/Aula%2014/Quick_final.html (acessado em 30/04/2004)

[4] - Vera Alves - Universidade do Vale do Rio dos Sinos - http://inf.unisinos.br/~vera/aula13.rtf (acessado em 29/04/2004)

[5] - Marinho Barcellos - - Universidade do Vale do Rio dos Sinos - http://www.inf.unisinos.br/~marinho/Teaching/Graduacao/Lab2/Aulas/09/09-abp.pdf (acessado em 29/04/2004)

[6] - Ramanan, Prakash – VLDB Conference - "Covering Indexes for XML Queries: Bisimulation Simulation = Negation" http://www.vldb.org/conf/2003/papers/S06P03.pdf

[7] - CHAN Chee Yong – SOC1 (chancy@comp.nus.edu.sg) - XML Topics - http://www.comp.nus.edu.sg/~chancy/ (acessado em 01/05/2004)

[8] - H. Wang, S. Park, W. Fan, P.S. Yu – SIGMOD 2003 - ViST: A dynamic index method for querying XML data by tree structures - http://www.comp.nus.edu.sg/~cs6203/L6.pdf (acessado em 01/05/2004)

[9] - H. Jiang, E. Wang, H. Lu, J.X. Yu - VLDB 2003 - Holistic twig joins on indexed XMLdocuments - http://www.comp.nus.edu.sg/~cs6203/L5b.pdf (acessado em 29/05/2004)

[10] - Bruno V. Rezende, Marcello P. Bax - Projeto Indexa: ferramentas de auxilia à divulgação de informações na Web - Universidade Federal de Minas Gerais http://cuba.paradigma.com.br/paradigma/artigos/artigos_04.pdf - (acessado em 29/05/2004)

[11] - Marcilio S. Chaves - Padrões em Bibliotecas Digitais – Universidade Católica do Rio Grande do Sul - http://www.dcc.ufmg.br/pos/html/spg2000/anais/hermes/hermes.htm - (acessado em 29/05/2004)

[12] - Ricardo Annes - PUCRS - http://pucrs.campus2.br/~annes/ - (acessado em 28/05/2004)

[13] - Cohen, S. , Mamou, J., Kanza, Y., Sagiv,Y. - "XSEarch: A Semantic Search Engine for XML" - VLDB 2003 - http://www.vldb.org/conf/2003/papers/S03P02.pdf (acessado em 01/05/2004)

[14] - H. Jiang, E. Wang, H. Lu, J.X. Yu , Beng Chin Ooi - XR-Tree: Indexing XML Data for Efficient Structural Joins – http://www.comp.nus.edu.sg/~ooibc/373-xrtree.pdf (acessado em 02/05/2004)

[15] - Edith Cohen, Haim Kaplan, and Tova Milo. Labeling dynamic XML trees. In PODS, págs 271-281, 2002.