

**UNIVERSIDADE FEDERAL DA PARAÍBA**  
**CENTRO DE CIÊNCIAS EXATAS E DA NATUREZA**  
**DEPARTAMENTO DE INFORMÁTICA**  
**PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA**

**Uma Técnica de Indexação de Dados Semi-  
Estruturados para o Processamento Eficiente de  
Consultas com Ramificação**

**Talles Brito Viana**

**João Pessoa - PB**  
**Março de 2011**

**Talles Brito Viana**

**Uma Técnica de Indexação de Dados Semi-Estruturados para o  
Processamento Eficiente de Consultas com Ramificação**

Dissertação de Mestrado submetida ao Programa de  
Pós-Graduação em Informática da Universidade  
Federal da Paraíba como requisito parcial para  
obtenção do título de Mestre em Informática.

**Orientador: Prof. Dr. Glêdson Elias da Silveira**

**João Pessoa - PB**

**Março de 2011**

V614u Viana, Talles Brito.

Uma técnica de indexação de dados semi-estruturados para o processamento eficiente de consultas com ramificação/ Talles Brito Viana.-- João Pessoa, 2011.

130f.

Orientador: Glêdson Elias da Silveira

Dissertação (Mestrado) – UFPB/CCEN

1. Informática. 2. Indexação. 3. Recuperação de informação.  
4. Dados semi-estruturados. 5. XML.

*UFPB/BC*

*CDU: 004(043)*

1

Ata da Sessão Pública de Defesa de Dissertação de Mestrado do TALLEs BRITO VIANA, candidato ao Título de Mestre em Informática na Área de Sistemas de Computação, realizada em 18 de marco de 2011.

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

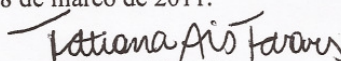
26

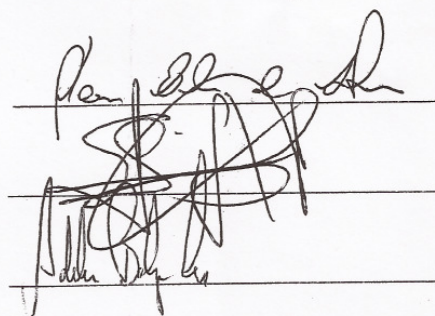
27

Prof. Dr. Glêdson Elias da Silveira  
Primeiro Orientador (DI-UFPB)

Prof. Dr. Guido Lemos de Souza Filho  
Examinador Interno (PPGI-UFPB)

Prof. Dr. Adilson Barboza Lopes  
Examinador Externo (UFRN)

  
Tatiana Aires Tavares



27

## Agradecimentos

Aos meus pais José Viana e Maria Socorro Brito por me darem forças nos momentos difíceis.

Por sempre estarem preocupadas comigo mesmo quando eu acho que está tudo bem: Thais Brito Viana e Viviane Brito Viana.

Aos meus amigos colegas de curso que sempre me ajudaram sem nenhuma má vontade e que sempre compartilhamos os bons momentos: Aluísio Bruno, Antônio Gomes, Arthur Neves, Caio Caroca, Claudio Romero, Gustavo Rezende, Herbet Ferreira, Leandro Baima, Max Porfírio, Ricardo Serrano, Ricardo Barbosa.

Aos amigos de laboratório por deixar o dia-a-dia mais divertido: Adriana Esmeraldo, Bruno Luna, Daniel Charles Porto, Gustavo Cavalcanti (*in memoriam*), João Paulo Oliveira, Sebastião Rabelo, Thaís Burity, Vinícius Souza, Yuri Morais.

Aos alunos de iniciação científica que contribuíram no desenvolvimento deste trabalho e aprenderam juntamente comigo vários tópicos interessantes: Hugo Nóbrega, Jailton Maciel, Marcos Paulo, Thiago Ribeiro.

Ao professor orientador pelo incentivo ao trabalho e pelas lições aprendidas: Prof<sup>o</sup> Glêdson Elias.

Aos professores da banca avaliadora pelas contribuições ao trabalho desenvolvido: Prof<sup>o</sup> Adilson Barboza, Prof<sup>o</sup> Guido Lemos.

Aos professores de coordenação e chefes de departamento pelo apoio, água, café, cadeira emprestada e ar condicionado: Prof<sup>o</sup> Hamilton Soares, Prof<sup>a</sup> Tatiana Aires, Prof<sup>a</sup> Valéria Gonçalves.

Um sincero obrigado!

## Resumo

O surgimento de sistemas baseados na Web tem gerado uma vasta quantidade de fontes de documentos semi-estruturados, os quais necessitam ser indexados por sistemas de busca a fim de possibilitar a descoberta de documentos de acordo com necessidades de informação do usuário. Entretanto, um dos maiores desafios no desenvolvimento de técnicas de indexação para documentos semi-estruturados diz respeito a como indexar não somente o conteúdo textual, mas também a informação estrutural dos documentos. O principal problema está em prover suporte para consultas com ramificação sem introduzir fatores que causem perda de precisão aos resultados de pesquisa, bem como, o crescimento indesejado do tempo de processamento de consultas e dos tamanhos de índice. Várias técnicas de indexação para dados semi-estruturados são encontradas na literatura. Apesar das relevantes contribuições, as propostas existentes sofrem com problemas relacionados à perda de precisão, requisitos de armazenamento ou custos de processamento de consultas. Neste contexto, nesta dissertação é proposta uma técnica de indexação (denominada BranchGuide) para dados semi-estruturados que suporta uma bem definida classe de consultas com ramificação sem perda de precisão. Esta classe compreende caminhos com ramificação que permitem expressar dependências pai-filho entre elementos nos quais podem ser impostas restrições sob os valores de atributos de tais elementos. Como evidenciado experimentalmente, a adoção da técnica BranchGuide gera excelentes tempos de processamento de consulta e tamanhos de índice menores do que os gerados por uma técnica de interseção estrutural.

**Palavras-chave:** Informática, Indexação, Recuperação de Informação, Dados Semi-Estruturados, XML.

# Abstract

The explosive growth of web-based information systems has created various sources and vast quantities of semi-structured data, which need to be indexed by search engines in order to allow the retrieval of documents according to user needs. However, one of the major challenges in the development of indexing techniques for semi-structured data is related to how to index not only textual but also structural content. The main issue is how to efficiently handle branching path expressions without introducing precision loss as well as undesired growth of query processing costs and index file sizes. Several proposals for indexing semi-structured data can be found in the literature. Despite their relevant contributions, existing proposals suffer from at least one of the problems related to precision loss, storage space requirements and query processing costs. In such a context, this thesis proposes an efficient, lossless path-based indexing technique (named as BranchGuide) for semi-structured data, which deals with a well-defined class of branching path expressions. This well-defined class includes branching paths that allow expressing parent-child dependencies between elements in which may be imposed restrictions over the textual value of attributes of such elements. As evinced by experimental evaluation, the adoption of the BranchGuide technique results in excellent query processing time and generates smaller index file sizes than a structural join indexing technique.

**Keywords:** Data Processing, Indexing Techniques, Information Retrieval, Semi-Structured Data, XML.

# Sumário

|                                                                               |             |
|-------------------------------------------------------------------------------|-------------|
| <b>AGRADECIMENTOS</b>                                                         | <b>V</b>    |
| <b>RESUMO</b>                                                                 | <b>VI</b>   |
| <b>ABSTRACT</b>                                                               | <b>VII</b>  |
| <b>SUMÁRIO</b>                                                                | <b>VIII</b> |
| <b>LISTA DE FIGURAS</b>                                                       | <b>X</b>    |
| <b>LISTA DE TABELAS</b>                                                       | <b>XIII</b> |
| <b>LISTA DE SIGLAS</b>                                                        | <b>XIV</b>  |
| <b>1 INTRODUÇÃO</b>                                                           | <b>1</b>    |
| 1.1 MOTIVAÇÃO                                                                 | 3           |
| 1.2 OBJETIVOS                                                                 | 4           |
| 1.3 CENÁRIO DE APLICAÇÃO                                                      | 6           |
| 1.4 CONTRIBUIÇÕES                                                             | 7           |
| 1.5 ORGANIZAÇÃO DO TRABALHO                                                   | 7           |
| <b>2 TRABALHOS RELACIONADOS</b>                                               | <b>10</b>   |
| 2.1 ÍNDICES BASEADOS EM CAMINHOS                                              | 10          |
| 2.1.1 <i>Dataguide</i>                                                        | 13          |
| 2.1.2 <i>O Problema do Processamento de Consultas com Ramificação</i>         | 15          |
| 2.2 ÍNDICES DE INTERSEÇÃO ESTRUTURAL                                          | 17          |
| 2.2.1 <i>eXist</i>                                                            | 18          |
| 2.2.2 <i>O Problema do Custo de Interseções</i>                               | 21          |
| 2.3 ÍNDICES DE SEQUÊNCIA                                                      | 22          |
| 2.3.1 <i>ViST</i>                                                             | 22          |
| 2.3.2 <i>O Problema de Filtragem de Falsos Positivos</i>                      | 23          |
| 2.4 CONSIDERAÇÕES FINAIS                                                      | 28          |
| <b>3 A TÉCNICA BRANCHGUIDE</b>                                                | <b>31</b>   |
| 3.1 PADRÃO DE SELEÇÃO DE CAMINHOS                                             | 34          |
| 3.2 INDEXAÇÃO                                                                 | 35          |
| 3.3 RECUPERAÇÃO DE INFORMAÇÃO                                                 | 42          |
| 3.4 CONSIDERAÇÕES FINAIS                                                      | 44          |
| <b>4 PROJETO DA TÉCNICA BRANCHGUIDE</b>                                       | <b>46</b>   |
| 4.1 PROJETO DO MECANISMO DE PROCESSAMENTO DE CONSULTAS                        | 47          |
| 4.1.1 <i>Método de Identificação de Atributos Irmãos</i>                      | 48          |
| 4.2 PROJETO DO ALGORITMO DE INDEXAÇÃO                                         | 51          |
| 4.2.1 <i>Duplicação de Nós</i>                                                | 51          |
| 4.2.2 <i>Mapeamento de Códigos de Rótulo-Valor</i>                            | 54          |
| 4.2.3 <i>Codificação de Caminhos como Sequências Numéricas</i>                | 55          |
| 4.2.4 <i>Tratamento da Replicação de Valores de Rótulo e Valores Textuais</i> | 56          |

|          |                                                                                    |            |
|----------|------------------------------------------------------------------------------------|------------|
| 4.2.5    | <i>Tratamento da Replicação de Identificadores de Documentos e Códigos de Eixo</i> | 57         |
| 4.2.5.1  | Organização de Chaves de Índice                                                    | 58         |
| 4.2.5.2  | Organização de Resultados de Índice                                                | 59         |
| 4.3      | CONSIDERAÇÕES FINAIS                                                               | 61         |
| <b>5</b> | <b>IMPLEMENTAÇÃO</b>                                                               | <b>63</b>  |
| 5.1      | IMPLEMENTAÇÃO DO MÉTODO DE ORGANIZAÇÃO DE CHAVES DE ÍNDICE                         | 64         |
| 5.1.1    | <i>A Função de hash perfeito</i>                                                   | 64         |
| 5.1.2    | <i>Ordenação de Chaves de Índice</i>                                               | 67         |
| 5.2      | IMPLEMENTAÇÃO DO ÍNDICE                                                            | 70         |
| 5.2.1    | <i>Processo de Indexação</i>                                                       | 70         |
| 5.2.2    | <i>Processo de Consulta ao Índice</i>                                              | 73         |
| 5.2.3    | <i>Organização do Índice de Caminhos</i>                                           | 76         |
| 5.2.4    | <i>Compressão de Índice</i>                                                        | 77         |
| 5.3      | CONSIDERAÇÕES FINAIS                                                               | 80         |
| <b>6</b> | <b>AVALIAÇÃO EXPERIMENTAL</b>                                                      | <b>83</b>  |
| 6.1      | AMOSTRAS PARA AVALIAÇÃO                                                            | 84         |
| 6.1.1    | <i>DBLP</i>                                                                        | 84         |
| 6.1.2    | <i>Amostra X-ARM</i>                                                               | 84         |
| 6.2      | CONFIGURAÇÃO DO EXPERIMENTO                                                        | 87         |
| 6.3      | EXPERIMENTOS COM A AMOSTRA DBLP                                                    | 88         |
| 6.3.1    | <i>Tamanho de Índice</i>                                                           | 88         |
| 6.3.2    | <i>Tempo de Execução de Consultas</i>                                              | 89         |
| 6.3.3    | <i>Tempo de Indexação</i>                                                          | 91         |
| 6.4      | EXPERIMENTOS COM A AMOSTRA X-ARM                                                   | 92         |
| 6.4.1    | <i>Tamanho de Índice</i>                                                           | 92         |
| 6.4.2    | <i>Tempo de Execução de Consultas</i>                                              | 92         |
| 6.4.3    | <i>Tempo de Indexação</i>                                                          | 98         |
| 6.5      | CONSIDERAÇÕES FINAIS                                                               | 99         |
| <b>7</b> | <b>CONSIDERAÇÕES FINAIS</b>                                                        | <b>100</b> |
| 7.1      | CONTRIBUIÇÕES                                                                      | 102        |
| 7.2      | BENEFÍCIOS                                                                         | 104        |
| 7.3      | LIMITAÇÕES                                                                         | 106        |
| 7.4      | TRABALHOS FUTUROS                                                                  | 107        |
|          | <b>REFERÊNCIAS</b>                                                                 | <b>109</b> |
|          | <b>APÊNDICE</b>                                                                    | <b>112</b> |

## Lista de Figuras

|                                                                                                    |    |
|----------------------------------------------------------------------------------------------------|----|
| Figura 1: Arquitetura genérica de um sistema de busca .....                                        | 6  |
| Figura 2. Documentos XML descrevendo livros publicados por editoras.....                           | 11 |
| Figura 3. Exemplo de grafo de dados .....                                                          | 11 |
| Figura 4. Caminhos replicados destacados .....                                                     | 12 |
| Figura 5. Exemplo de índice Dataguide.....                                                         | 14 |
| Figura 6: Expressões com ramificação e decomposição em expressões simples.....                     | 15 |
| Figura 7: Perda de informações após indexação .....                                                | 16 |
| Figura 8: Exemplo de índice gerado pelo eXist .....                                                | 19 |
| Figura 9: Processamento de consulta utilizando interseção estrutural.....                          | 20 |
| Figura 10: Sequências $s_1$ e $s_2$ geradas a partir dos documentos $d_1$ e $d_2$ .....            | 23 |
| Figura 11: Árvore de sufixos $T_{sf}$ após inserção das sequências $s_1$ e $s_2$ da Figura 10..... | 24 |
| Figura 12: Alteração de ordem de elementos na geração de sequência.....                            | 26 |
| Figura 13: Geração da sequência $S_q$ a partir de informações da consulta .....                    | 26 |
| Figura 14: Comparação da sequência $S_q$ com as sequências $S_D$ e $S'_D$ .....                    | 27 |
| Figura 15: Regras de cardinalidade .....                                                           | 34 |
| Figura 16: Padrão de seleção de caminhos proposto .....                                            | 35 |
| Figura 17: Exemplos de caminhos selecionados .....                                                 | 35 |
| Figura 18: Um exemplo de fonte de dados .....                                                      | 37 |
| Figura 19: $G'$ após aplicação da regra I .....                                                    | 37 |
| Figura 20: $G'$ após aplicação da regra II.....                                                    | 38 |
| Figura 21: Regra III .....                                                                         | 38 |
| Figura 22: $G'$ após primeira aplicação da regra III.....                                          | 39 |
| Figura 23: $G'$ após segunda aplicação da regra III .....                                          | 40 |
| Figura 24: Caminhos extraídos a partir da fonte de dados.....                                      | 41 |
| Figura 25: Controle de similaridade através de operadores .....                                    | 43 |
| Figura 26. Decomposição de expressão de consulta.....                                              | 44 |
| Figura 27: Relação entre Dataguide e BranchGuide.....                                              | 44 |

|                                                                               |     |
|-------------------------------------------------------------------------------|-----|
| Figura 28: Checagem de eixo de atributo irmão.....                            | 47  |
| Figura 29: Exemplo de atribuição de códigos de eixo de atributo irmão .....   | 49  |
| Figura 30: Documento de exemplo 3 .....                                       | 52  |
| Figura 31: Criando subárvores através da Regra III .....                      | 52  |
| Figura 32: Grafo de índice $G'$ obtido após indexação sob o Documento 3 ..... | 53  |
| Figura 33: Atribuição de códigos rótulo-valor .....                           | 54  |
| Figura 34: Caminho representado através de código de rótulo-valor .....       | 55  |
| Figura 35: Índice invertido de caminhos.....                                  | 57  |
| Figura 36: Resultados replicados.....                                         | 58  |
| Figura 37: Exemplo de documento com códigos de eixo de atributo irmão.....    | 60  |
| Figura 38: Corte de vértices duplicados.....                                  | 62  |
| Figura 39: Equação do conjunto de <i>base</i> .....                           | 65  |
| Figura 40: Função de hash perfeito $h$ .....                                  | 66  |
| Figura 41: Arquitetura do índice.....                                         | 70  |
| Figura 42: Processo de indexação.....                                         | 71  |
| Figura 43: Processamento de uma consulta .....                                | 74  |
| Figura 44: Organização do índice de caminhos.....                             | 76  |
| Figura 45: Aplicação de compressão por prefixo .....                          | 78  |
| Figura 46: Aplicação de compressão por intervalo .....                        | 78  |
| Figura 47: Valores de endereço armazenados.....                               | 79  |
| Figura 48: Relações de reuso geradas na amostra X-ARM .....                   | 85  |
| Figura 49: Tamanho de índice: DBLP .....                                      | 88  |
| Figura 50: Comparação de tempo de execução da consulta $Q_1$ .....            | 89  |
| Figura 51: Comparação de tempo de execução da consulta $Q_2$ .....            | 90  |
| Figura 52: Tempo de indexação: DBLP.....                                      | 91  |
| Figura 53: Tamanho de índice: X-ARM .....                                     | 92  |
| Figura 54: Comparação de tempo de execução da consulta $Q_3$ .....            | 94  |
| Figura 55: Comparação de tempo de execução da consulta $Q_4$ .....            | 94  |
| Figura 56: Comparação de tempo de execução da consulta $Q_5$ .....            | 94  |
| Figura 57: Comparação de tempo de execução da consulta $Q_6$ .....            | 95  |
| Figura 58: Comparação de tempo de execução da consulta $Q_7$ .....            | 97  |
| Figura 59: Comparação de tempo de execução da consulta $Q_8$ .....            | 97  |
| Figura 60: Tempo de indexação: X-ARM.....                                     | 98  |
| Figura 61: Representação de um elemento .....                                 | 112 |
| Figura 62: Representação de um atributo.....                                  | 113 |

|                                                                  |     |
|------------------------------------------------------------------|-----|
| Figura 63: Hierarquia entre elementos.....                       | 113 |
| Figura 64: Atributo de um elemento.....                          | 113 |
| Figura 65: Exemplo de Documento XML e grafo correspondente ..... | 114 |
| Figura 66: Nós alvo de consultas XPath. ....                     | 115 |
| Figura 67: Eixos do XPath .....                                  | 116 |

## Lista de Tabelas

|                                                                   |    |
|-------------------------------------------------------------------|----|
| Tabela 1: Tabela comparativa.....                                 | 28 |
| Tabela 2: Caminhos e sequências numéricas .....                   | 56 |
| Tabela 3: Tabela de mapeamento de códigos para a altura 3 .....   | 57 |
| Tabela 4: Ordenação de chaves de caminhos para a altura 3 .....   | 59 |
| Tabela 5: Indexação do Documento 3.....                           | 60 |
| Tabela 6: Caminhos com ramificação reusados .....                 | 61 |
| Tabela 7: Conjunto de tuplas representantes .....                 | 65 |
| Tabela 8: Mapeamento perfeito .....                               | 66 |
| Tabela 9: Caminhos codificados e tuplas correspondentes .....     | 67 |
| Tabela 10: Inversão de tuplas e aplicação da função de hash ..... | 69 |
| Tabela 11: Resultado final da ordenação .....                     | 69 |
| Tabela 12: Propriedades da amostra DBLP .....                     | 84 |
| Tabela 13: Propriedades da amostra X-ARM .....                    | 87 |
| Tabela 14: Fatores e níveis.....                                  | 87 |
| Tabela 15: Consultas para a amostra DBLP.....                     | 89 |
| Tabela 16: Tempos de consulta para $Q_1$ .....                    | 90 |
| Tabela 17: Tempos de consulta para $Q_2$ .....                    | 91 |
| Tabela 18: Consultas para a amostra X-ARM.....                    | 93 |
| Tabela 19: Tempos de consulta para $Q_3$ .....                    | 95 |
| Tabela 20: Tempos de consulta para $Q_4$ .....                    | 95 |
| Tabela 21: Tempos de consulta para $Q_5$ .....                    | 96 |
| Tabela 22: Tempos de consulta para $Q_6$ .....                    | 96 |
| Tabela 23: Tempos de consulta para $Q_7$ .....                    | 97 |
| Tabela 24: Tempos de consulta para $Q_8$ .....                    | 98 |

## Lista de Siglas

DBLP – Digital Bibliography & Library Project

NEXI – Narrowed Extended XPath

ViST – Virtual Suffix Tree

Web – World Wide Web

XISS – XML Indexing & Storage System

XML – Extensible Markup Language

XPath – XML Path Language

# 1 Introdução

Nas técnicas clássicas de recuperação de informação, os documentos são vistos como uma coleção não estruturada de termos textuais [Weigel02]. Entretanto, o que acontece na realidade é que dificilmente os documentos são de fato completamente não estruturados [Manning08]. Por exemplo, geralmente, um livro é composto de capítulos e seções. Na verdade, as relações estruturais entre elementos de dados dos documentos podem expressar conteúdo semântico relacionado ao domínio de aplicação. Este tipo de modelo de representação de dados é classificado como *semi-estruturado*, no qual a representação semântica dos dados (definida pela estrutura dos documentos) é incorporada juntamente com os valores textuais. Neste sentido, vários modelos para a representação e armazenamento de dados têm sido propostos. Por exemplo, o modelo de representação XML (*Extensible Markup Language*) [W3C07] tem sido razoavelmente utilizado em diversas aplicações e tem se tornado um padrão para troca de informações na *Web* (*World Wide Web*) [Li01].

Assim, a adoção de técnicas para tratar eficientemente a indexação e recuperação de dados semi-estruturados é essencial para que usuários de sistemas de busca possam construir consultas levando em consideração relações estruturais tanto quanto textuais. Apesar disto, é importante ressaltar que os principais desafios envolvidos no desenvolvimento de técnicas para indexação de dados semi-estruturados estão relacionados com a indexação das propriedades estruturais dos documentos e não somente informações textuais, pois preservar as propriedades estruturais é bastante custoso em termos de espaço de armazenamento dos índices. Além disso, as consultas submetidas pelos usuários ao sistema de busca podem ter restrições textuais, bem como estruturais, implicando em um considerável aumento no tempo de processamento das consultas, como relatado em [Catania05]. Neste sentido, em resumo, o principal problema a ser tratado no desenvolvimento de uma técnica de recuperação de dados semi-estruturados é prover a indexação dos dados sem ignorar as relações estruturais dos

documentos, ao mesmo tempo em que se provê uma recuperação eficiente em termos de tempo de processamento.

Diversas técnicas de indexação de dados semi-estruturados têm sido propostas para lidar com tais problemas. A fim de reduzir o tamanho dos arquivos índice, as técnicas de indexação classificadas como *baseadas em caminhos* indexam somente um subconjunto de caminhos selecionados (um caminho representa uma sequência de navegação através de elementos e atributos sob a estrutura de um documento XML). Em tais técnicas, somente os caminhos que satisfazem um determinado padrão são selecionados para a indexação, consequentemente, indexando somente um subconjunto de todos os possíveis caminhos que poderiam ser extraídos da fonte de dados.

Como um exemplo clássico, a técnica *Dataguide* [Goldman97] indexa somente *expressões de caminho simples*. Uma expressão de caminho simples representa uma sequência de navegação que corresponde a uma cadeia linear (isto é, sem bifurcações) sob a estrutura do documento XML. Embora o *Dataguide* reduza os tamanhos dos índices, este não pode lidar adequadamente com *expressões de consulta com ramificação* (abreviadamente *consultas com ramificação*). Uma expressão de consulta com ramificação representa uma sequência de navegação que contém ramificações (bifurcações), correspondendo a uma pequena árvore sob a estrutura do documento XML.

Com o objetivo principal de tratar limitações funcionais das técnicas baseadas em caminhos, tal como o suporte ao processamento de consultas com ramificação, outras técnicas de indexação propostas (como, por exemplo, em [Li01] e [Meier02]), denominadas de *índices de interseção estrutural* não fazem uso de padrões de caminhos para selecionar caminhos para indexação. Ao invés disto, as técnicas de interseção estrutural indexam diretamente os nomes dos elementos, atributos e valores textuais dos documentos XML. Mais especificamente, estas técnicas empregam um tipo de algoritmo comumente categorizado como *interseção estrutural*, no qual cada consulta é decomposta em vários nomes (de elementos/atributos) individuais, de forma que uma consulta ao índice é executada para cada nome individual a fim de obter o resultado associado. Então, o processador de consultas combina todos os resultados individuais obtidos através de um algoritmo de interseção estrutural a fim de obter o resultado final da execução da consulta.

Ao contrário das técnicas de indexação baseadas em caminhos (tal como o *Dataguide*), as técnicas baseadas em interseção estrutural podem processar consultas com ramificação uma vez que qualquer consulta com ramificação (que pode ser vista como uma árvore) pode ser

decomposta em partes menores, especificamente, em nomes individuais (que podem ser vistos como os nós individuais da árvore), que por sua vez, podem ser processados e posteriormente combinados. Apesar destes benefícios, operações de interseção estrutural têm alto custo de processamento, e, como demonstrado experimentalmente em [Kaushik04], a melhoria de desempenho no processamento de consultas depende da redução da quantidade de operações de interseção estrutural.

## 1.1 Motivação

Uma perda de precisão inerente ocorre quando dados semi-estruturados são indexados utilizando técnicas clássicas de indexação textual. Isto acontece porque as relações estruturais entre elementos XML não são mantidas após indexação, de forma que não é possível prover um processamento de consultas que considere restrições estruturais. Além disso, preservar relações estruturais após indexação é bastante custoso do ponto de vista de armazenamento e processamento, logo, a principal motivação é como prover um mecanismo de indexação e recuperação que trate eficientemente expressões de consulta com restrições estruturais. Em outras palavras, como prover técnicas de indexação para XML com recuperação eficiente e sem alto custo de armazenamento, mas sem abrir mão de requisitos funcionais como o processamento de consultas com restrições estruturais.

Índices baseados em caminhos simples (tal como o *Dataguide* [Goldman97]) suportam somente expressões de caminhos simples. Estas técnicas são convenientes para processar somente consultas com única condição de restrição apenas no final da expressão. Assim, a fim de prover suporte para o processamento de consultas com ramificação é necessário projetar técnicas de indexação que obrigatoriamente indexam relações estruturais suficientes para prover tal funcionalidade. O emprego de técnicas de indexação que suportem o processamento de consultas com ramificação além de consultas textuais e de caminhos simples em sistemas de recuperação deverá prover um aumento da precisão total dos sistemas ao minimizar a quantidade de informação estrutural descartada após indexação.

Já os índices de interseção estrutural (tal como o *XISS* [Li01]) mantêm todas as propriedades estruturais existentes nos documentos, permitindo o processamento de expressões de caminhos simples e de consultas com ramificação, mas em compensação, o processamento de consultas é custoso [Kaushik04]. Logo, para obter um mecanismo escalável de processamento de consultas com ramificação deve ser evitado o uso de operações de

interseção estrutural, pois, em relação ao desempenho, a utilização de operações de interseção estrutural torna inadequado o emprego da indexação de fontes de dados XML de larga escala.

Neste contexto, em resumo, a motivação fundamental do presente trabalho está em como desenvolver uma técnica de indexação de dados semi-estruturados que suporte o processamento de consultas com ramificação, ao mesmo tempo em que, proveja tempos eficientes de processamento de consultas, bem como, tamanhos de índice de baixo custo, de forma que o mecanismo de indexação possa ser empregado em cenários de larga-escala.

## 1.2 Objetivos

Com o objetivo de prover um mecanismo de indexação e processamento de consultas com ramificação que gere tempos de processamento de consulta eficientes e índice com baixo custo de armazenamento, nesta dissertação é proposta uma técnica de indexação de dados semi-estruturados baseada em caminhos denominada *BranchGuide*. A técnica proposta é concebida para manipular uma definida classe de expressões de ramificações que é mais expressiva do que a suportada por índices de caminhos simples. A técnica BranchGuide pode ser empregada em sistemas de recuperação de dados semi-estruturados que adotam o padrão de descrição de dados XML.

É importante ressaltar que a técnica de indexação BranchGuide evita o uso de operações de interseção estrutural, e mesmo sem fazer uso deste recurso, não retorna falsos positivos no processamento da classe de consultas com ramificação suportada. Isto ocorre devido ao fato que o índice gerado mantém informações estruturais suficientes para processar a classe de consultas com ramificação suportada, mas sem gerar elevada sobrecarga no armazenamento do índice como efeito colateral. Desta forma, é possível alcançar um maior nível de precisão, mas sem causar impacto negativo relevante nos tempos de processamento de consulta e tamanhos de índice gerados.

Especificamente, BranchGuide é uma técnica de indexação exata (sem erros de precisão) para uma classe de expressões de caminhos com ramificação projetada de acordo com as seguintes características:

- **Adoção de um Padrão de Seleção de Caminhos:** Adoção de um padrão bem definido para seleção de expressões com ramificação evitando assim o retorno de falsos positivos no processo de consulta por expressões que pertencem a tal classe. São selecionados para indexação todos os caminhos simples e um conjunto substancial de caminhos com ramificação a partir dos documentos da fonte de dados. Sendo

assim, BranchGuide indexa caminhos mais expressivos do que técnicas de indexação de caminhos simples.

- **Recuperação de Documentos:** Adoção da unidade de recuperação definida como o documento ao invés de particulares elementos e atributos XML. O objetivo é recuperar um objeto que é descrito por um documento XML.
- **Processamento Eficiente de Consultas:** Suporte ao processamento eficiente de consultas com ramificação de forma que se tenha um desempenho superior às técnicas de interseção estrutural.
- **Geração de Índices de Baixo Custo:** Geração de tamanhos de índice menores do que os obtidos na indexação de dados semi-estruturados utilizando técnicas de interseção estrutural.

A técnica BranchGuide adota um índice de mapeamento de caminhos com ramificação para identificadores de documentos. Os caminhos mapeados satisfazem um definido padrão de seleção de caminhos. A primeira questão a ser tratada neste trabalho é a definição do padrão de seleção utilizado, bem como, a definição de um método de extração e indexação de caminhos que estão em conformidade com tal padrão. Além disso, o sucesso da técnica BranchGuide depende de um projeto de armazenamento de índice que não introduza impactos negativos relevantes. Em outras palavras, o formato e método de armazenamento e leitura do índice não devem introduzir fatores de perda de precisão ou aumento no tempo de consulta.

Nesta dissertação, com o objetivo de ressaltar a relevância do trabalho proposto, através de uma tabela comparativa, trabalhos relacionados com a técnica BranchGuide são comparados e discutidos. Além disso, a fim de avaliar a técnica BranchGuide de forma quantitativa foram realizados experimentos. O objetivo geral dos experimentos é medir parâmetros de tempo de recuperação e tamanhos de índice gerados pela técnica BranchGuide e compará-los com uma técnica de indexação baseada em interseção estrutural.

Por fim, em linhas gerais, os objetivos específicos do trabalho proposto são listados:

- Obter uma revisão de literatura sobre técnicas de indexação e recuperação de dados semi-estruturados a fim de avaliar os requisitos de precisão e suporte a consultas com ramificação, tempo de resposta, e tamanhos de índices gerados.

- Definir e especificar o projeto e implementação de uma técnica de indexação e recuperação (denominada BranchGuide) de documentos semi-estruturados com suporte a consultas de ramificação e sem perda de precisão.
- Avaliar experimentalmente os requisitos de tempo de processamento de consultas e tamanhos de índice gerados pela técnica BranchGuide e comparar com técnicas de indexação baseadas em interseção estrutural.

### 1.3 Cenário de Aplicação

Um sistema de busca deve encontrar documentos dentre uma larga coleção de documentos [Manning08]. Neste sentido, a técnica BranchGuide pode ser empregada em sistemas de busca para documentos semi-estruturados. Uma arquitetura genérica de sistema de busca que adota a técnica BranchGuide é composta pelos componentes mostrados na Figura 1. Como apontado, uma *Fonte de Dados XML* é empregada para posterior indexação. De posse da fonte de documentos XML, o *Motor de Indexação* é responsável por indexá-la, empregando a técnica BranchGuide, e assim, gerando o *Índice*. Já o *Usuário* interage com o sistema de busca através de uma interface gráfica que o auxilia a construir consultas de acordo com as necessidades de informação do mesmo.

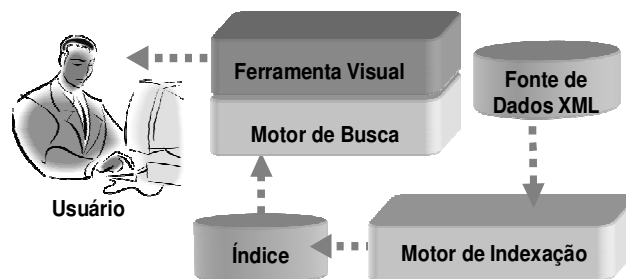


**Figura 1: Arquitetura genérica de um sistema de busca**

Uma vez que o Índice está disponível e dada uma consulta informada pelo usuário, o *Motor de Busca* processa a consulta a fim de recuperar documentos XML que estão de acordo com os critérios da consulta. É importante notar que enquanto o *Motor de Indexação* é responsável pela geração do índice, o *Motor de Busca* é responsável por consultar o índice. A técnica BranchGuide é incorporada a um sistema de busca para documentos semi-estruturados tanto no *Motor de Indexação* quanto no *Motor de Busca*.

O algoritmo de indexação da técnica BranchGuide recebe como entrada um conjunto de documentos XML e gera como saída um índice específico para ser consultado pelo algoritmo de busca da técnica BranchGuide. O algoritmo de busca da técnica BranchGuide recebe como

entrada expressões com ramificação que podem também ter operadores lógicos. Estas expressões são processadas e o índice é acessado de forma a gerar um resultado de pesquisa final. Como saída do algoritmo de busca é retornado um conjunto de identificadores de documentos que pertencem à fonte de dados. Com isto o usuário pode inspecionar (através de uma *Ferramenta Visual*) os documentos retornados.

## 1.4 Contribuições

Dentre as principais contribuições originadas a partir da proposição e concepção da técnica BranchGuide pode-se listar:

- A construção de uma nova técnica de indexação que provê suporte para uma classe de consultas com ramificação que permite processar, sem perda de precisão, expressões de consultas mais expressivas do que expressões de caminhos simples.
- Como evidenciado por uma avaliação experimental, ao contrário de técnicas de indexação baseadas em interseção estrutural, temos que a técnica BranchGuide não sofre do crescimento indesejado dos tempos de processamento de consultas e dos tamanhos de índice gerados.
- Como resultado do estudo desenvolvido, a fim de se obter um aprimoramento no tempo de processamento de consultas verifica-se que indexar uma grande quantidade de caminhos com ramificação é uma melhor direção a seguir do que usar operações de interseção estrutural para processar caminhos com ramificação.
- Por fim, como a técnica BranchGuide provê uma solução eficiente do ponto de vista dos requisitos de tempo de processamento e armazenamento, temos que em termos práticos, a proposição da técnica BranchGuide fomenta a construção e desenvolvimento de sistemas de busca de documentos em fontes de dados XML de larga escala sob cenários como a Web.

## 1.5 Organização do Trabalho

O restante desta dissertação está organizado da seguinte forma:

No Capítulo 2 são apresentados os trabalhos relacionados e motivações para o desenvolvimento da técnica BranchGuide. Em resumo, são discutidas três diferentes técnicas de indexação, cada uma pertencente a uma categoria diferente: índices de resumo, índices de interseção estrutural e índices de sequência. Cada uma destas categorias traz um determinado

problema. Este conjunto de problemas delinea a motivação para definição e projeto da técnica BranchGuide. Os problemas citados são referentes ao suporte de consultas com ramificação, mas sem a perda de precisão e sem gerar altos custos de armazenamento e processamento.

No Capítulo 3 a técnica BranchGuide é inicialmente delineada tanto em termos do algoritmo de indexação bem como em termos do algoritmo de busca propostos. A técnica BranchGuide adota um padrão de seleção de caminhos para indexação. Neste sentido, é apresentado um conjunto de motivações que conduzem para a adoção do uso de padrões de seleção de caminhos, bem como, em específico, as motivações que dão origem ao padrão adotado pela técnica. Além disso, é discutida a relação entre a adoção de um padrão de seleção de caminhos e o impacto que isto causa na definição dos requisitos funcionais e não funcionais de sistemas de busca que façam uso da técnica BranchGuide.

Por fim, definido o padrão de seleção de caminhos proposto, o Capítulo 3 apresenta a técnica de indexação que recebe como entrada uma fonte de dados XML e a partir disto deve extrair o conjunto de caminhos que satisfazem o padrão de seleção definido. Dado o cenário em que temos um conjunto de caminhos já indexados, o capítulo também apresenta como a técnica BranchGuide é empregada na construção de consultas em um sistema de busca.

Em geral, a quantidade de informações indexadas pela técnica BranchGuide é maior do a quantidade de informações armazenadas na fonte de dados já que a técnica indexa um conjunto amplo de expressões com ramificação. No Capítulo 4 são apontados os fatores que resultam nesse efeito não desejado.

Com o objetivo de tratar esse problema, na definição da técnica BranchGuide, é adotado um projeto de armazenamento de índice sem perda de precisão que reduz a quantidade informações armazenadas a fim de reduzir o custo de armazenamento. Neste sentido, no Capítulo 4 é apresentado, detalhado e discutido o projeto do algoritmo de indexação da técnica BranchGuide. Vale ressaltar que, além da especificação do método de organização dos dados indexados também é apresentado como que as estruturas de dados projetadas devem ser acessadas no momento da consulta ao índice.

No Capítulo 5 são tratadas as questões de implementação dos algoritmos de indexação e consulta da técnica de indexação BranchGuide. Primeiramente, uma função de ordenação de chaves de índice é discutida. O método de ordenação de chaves proposto visa respeitar os requisitos definidos no projeto do algoritmo de indexação da técnica BranchGuide. Na

sequência, a arquitetura do índice gerado pela técnica BranchGuide é descrita. Além disso, tendo como referência os módulos que compõem o índice, é discutido como estes componentes são gerados e/ou empregados durante o processo de indexação bem como no processo de consulta ao índice. Por fim, além das questões anteriormente citadas, no Capítulo 5 também são discutidas as técnicas de compressão de arquivos adotadas no armazenamento do índice.

No Capítulo 6 é delineado, apresentado e discutido um conjunto de experimentos empregando a técnica BranchGuide. Inicialmente são modeladas amostras com diferentes características para serem indexadas com a técnica BranchGuide e com um método de indexação baseado em interseção estrutural, o objetivo principal é comparar os tempos de consulta, tempos de indexação e tamanhos de índice gerados ao empregar as duas diferentes técnicas de indexação.

Por fim, no Capítulo 7 é apresentado um conjunto de considerações finais. São destacadas e enumeradas as principais contribuições trazidas com a proposição da técnica BranchGuide. Além disso, os benefícios do uso da técnica BranchGuide são listados e discutidos. Por fim, algumas limitações da técnica BranchGuide são apresentadas e direções de solução destas questões abertas são enumeradas como trabalhos futuros.

## 2 Trabalhos Relacionados

Neste capítulo, em ordem, são apresentados os índices baseado em caminhos, índices de interseção estrutural e, por fim, índices de sequência. Para cada categoria de índices são apresentados problemas relacionados e, de forma detalhada, as principais motivações para a criação da técnica de indexação proposta nesta dissertação. Por fim, através de uma tabela comparativa, de forma resumida, os trabalhos relacionados e o trabalho proposto são comparados e discutidos.

### 2.1 Índices Baseados em Caminhos

O modelo de dados XML é um baseado em hierarquia: cada *elemento* XML pode conter outros elementos filhos. Além disso, elementos XML podem conter *atributos* XML. Um atributo é um par nome-valor associado com um elemento em que informações sobre o respectivo elemento são armazenadas. Mais especificamente, os atributos fornecem informações adicionais que descrevem os elementos [Harold01].

Neste contexto, um documento XML é visto como uma árvore na qual os nós representam elementos ou atributos e as relações de hierarquia *pai*→*filho* contidas na árvore representam as relações existentes entre cada elemento e seus nós filhos no documento XML. Neste sentido, como definido por Weigel [Weigel02], comumente, ao considerarmos uma coleção de documentos XML (isto é, uma base de dados que contém mais do que um documento), todas as árvores correspondentes a cada documento desta coleção são concatenadas em um grafo definido como o *grafo de dados*. O grafo de dados representa a coleção como um todo através de um único grafo.

Por exemplo, a Figura 2 mostra dois documentos XML, cada um descreve os livros publicados por editoras. A primeira editora, descrita pelo *Documento 1* é denominada *SparkNotes*, que publica o livro *Hamlet*, cujo nome do autor é *Shakespeare*. A segunda

editora, descrita pelo *Documento 2* é denominada *Simon&Schuster*, que publica não somente o livro *Hamlet*, como também o livro *Othello*, ambos do autor *Shakesperare*.

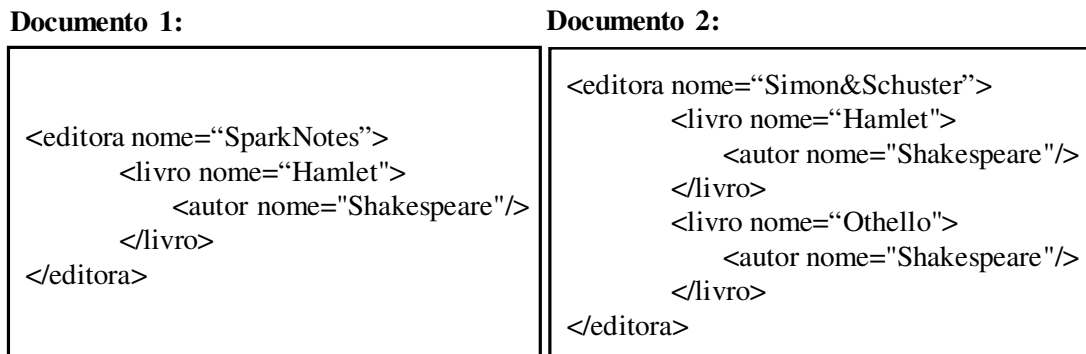


**Figura 2. Documentos XML descrevendo livros publicados por editoras**

Os dois documentos mostrados na Figura 2 são novamente apresentados utilizando o grafo de dados mostrado na Figura 3. A notação adotada para apresentar o grafo de dados é detalhada no Apêndice A.1. Um grafo de dados é um grafo direcionado em que o grafo de representação de cada documento da base de dados XML é concatenado a um vértice *RAIZ*. Cada vértice do grafo de dados está associado a algum elemento ou atributo dos documentos XML da base de dados, de forma que todos os caminhos que ocorrem na fonte de dados são representados no grafo de dados. Por fim, é importante ressaltar que as direções das arestas seguem a direção do relacionamento *pai*→*filho*.

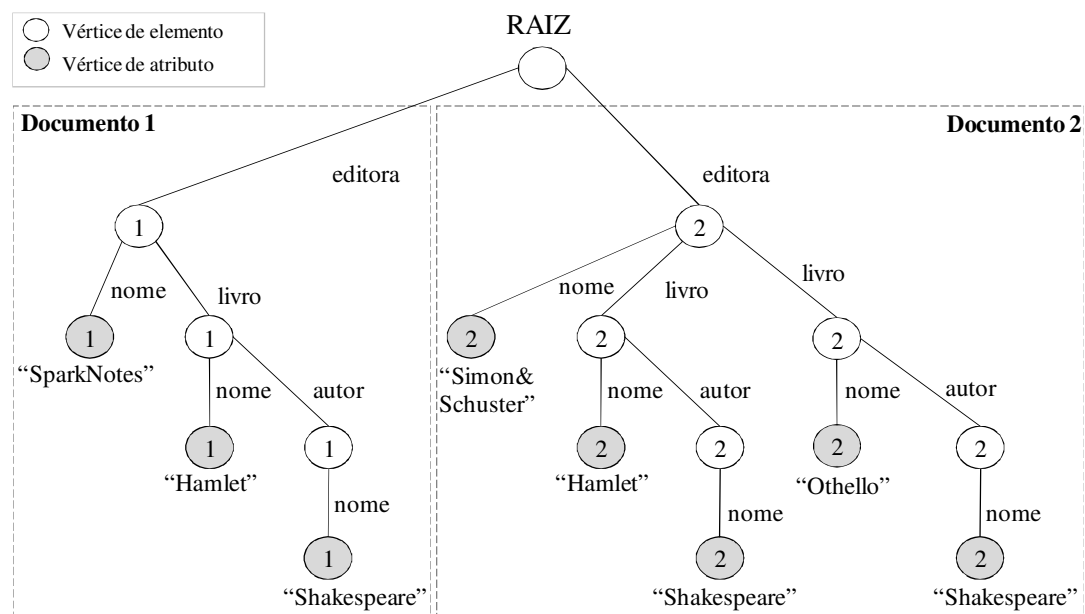


**Figura 3. Exemplo de grafo de dados**

Mais especificamente, considere o *Documento 1* da Figura 2. Este documento contém a seguinte relação no sentido de navegação *pai*→*filho* a partir do elemento raiz:

*editora–livro–autor–nome–“Shakespeare”*. Então, este caminho está contido no grafo de dados mostrado na Figura 4. Cada vértice que compõe o caminho contém o identificador de documento no qual existe a ocorrência do caminho.

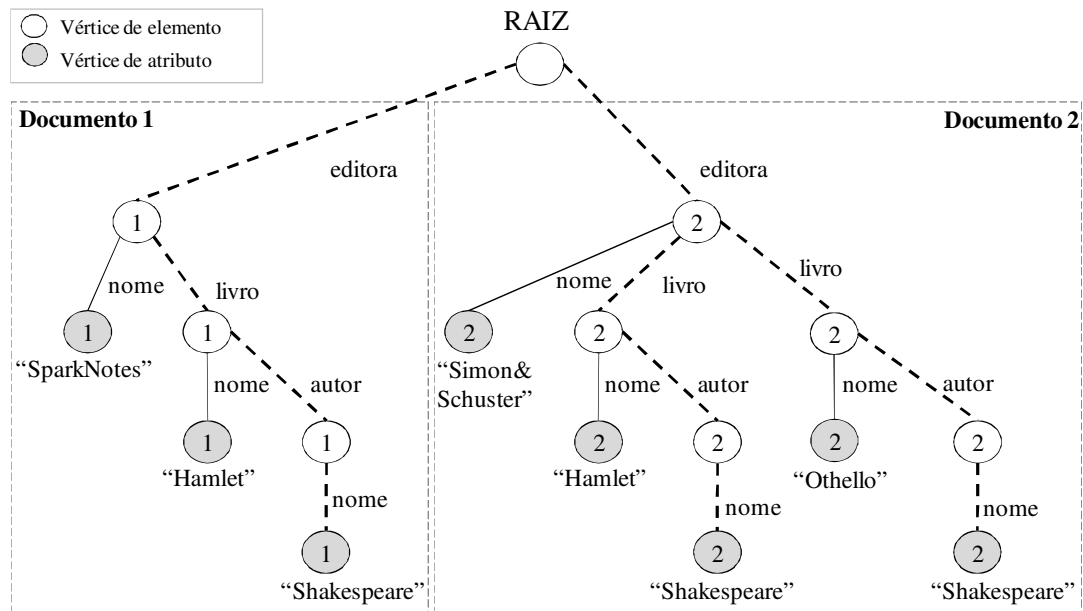


**Figura 4. Caminhos replicados destacados**

Conforme pode ser observado na Figura 4, o caminho de navegação *editora–livro–autor–nome–“Shakespeare”* ocorre nos *Documentos 1 e 2*, ressaltando que no *Documento 2* o caminho ocorre duas vezes. Cada uma das ocorrências do caminho *editora–livro–autor–nome–“Shakespeare”* nos *Documentos 1 e 2* é destacada na Figura 4 através de linhas pontilhadas. Logo, a partir deste exemplo podemos concluir que um mesmo caminho de navegação é replicado no grafo de dados quando vários documentos descrevem os mesmos dados.

Para o caminho de navegação *editora–livro–autor–nome–“Shakespeare”*, ao navegar em sentido *pai→filho* a partir da RAIZ sob o grafo de dados mostrado na Figura 4, a ocorrência replicada (ocorrendo três vezes) deste caminho no grafo de dados resulta na recuperação de dois diferentes identificadores de documentos, representados especificamente pelo conjunto união dos identificadores  $\{1\} \cup \{2\} = \{1, 2\}$ . Neste sentido, ao invés de procurar ocorrências de caminhos diretamente no grafo de dados pode-se adotar um procedimento otimizado de busca sob um índice em que não ocorrem caminhos replicados dentro do espaço de busca. Neste sentido, as técnicas de indexação para dados semi-estruturados evitam buscar por caminhos diretamente a partir da fonte de dados, mas buscar em uma estrutura compacta que represente toda a coleção de dados armazenados.

Para processar consultas de maneira eficiente, técnicas de indexação *baseadas em caminhos* geram um índice contendo um resumo dos documentos que compõem a fonte de dados [Weigel02]. A forma como um índice baseado em caminhos é criado pode ser definida através de regras específicas de uma determinada técnica de indexação. No momento da criação de tais resumos das informações presentes no grafo de dados, essas regras definem uma transformação do grafo de dados em um novo grafo, denominado de *grafo do índice*. Um processo de indexação pode ser definido como uma função  $I: G \rightarrow G'$  que mapeia um grafo de dados  $G$  para um grafo de índice  $G'$  de acordo com as regras definidas por  $I$ .

Em geral a função de transformação  $I$  reduz a quantidade de nós de  $G$ , mesclando os nós existentes no grafo de dados. Por esta razão, na literatura, os índices baseados em caminhos também são denominados de *índices de resumo*, já que é gerado um resumo dos dados de entrada utilizando heurísticas específicas do método de indexação. É importante ressaltar que pode ocorrer perda de informações na geração do grafo de índice, pois o resumo tende a sumarizar informações da fonte de dados. Um exemplo clássico de índice baseado em caminhos em que ocorre a perda de informações é o *Dataguide* [Goldman97], detalhado na subseção a seguir.

### 2.1.1 Dataguide

O Dataguide [Goldman97] é um índice baseado em caminhos que define a seguinte regra no processo de indexação: *Todo caminho simples existente do grafo de dados deve ser representado somente uma única vez no grafo de índice*. Neste caso, os vértices de um determinado caminho do grafo de índice referenciam o conjunto união de todos os documentos que contém o respectivo caminho no grafo de dados. Para o exemplo da Figura 4, o correspondente índice *Dataguide* é ilustrado na Figura 5. Na Figura 5, temos que os vértices dos caminhos de navegação do grafo de dados (Figura 4) são mesclados se existir o mesmo caminho em mais do que um documento no grafo de dados. Por exemplo, o caminho *editora–livro–autor–nome– “Shakespeare”* ocorre nos documentos 1 e 2, de forma que os vértices do caminho no grafo de índice referenciam o conjunto  $\{1, 2\}$ . Neste sentido, ao invés de existir a ocorrência de três caminhos *editora–livro–autor–nome– “Shakespeare”* replicados no grafo de dados, temos um único caminho no grafo de índice que indica diretamente os documentos da fonte de dados nos quais ocorre o caminho (este caminho do grafo de índice está destacado com uma linha pontilhada na Figura 5). Desta forma, o acesso direto a um determinado caminho no grafo índice, que nunca é replicado no grafo de índice, reduz o espaço de busca e aperfeiçoa o tempo de processamento de consultas.

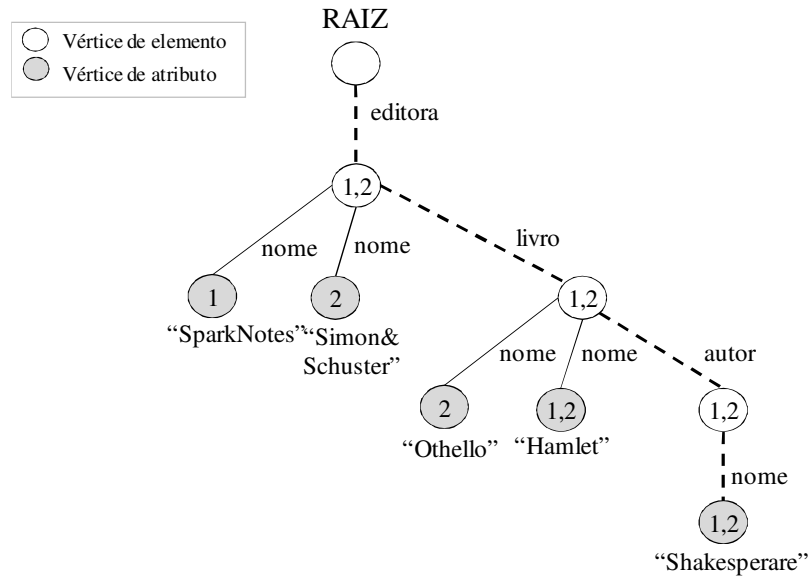


**Figura 5. Exemplo de índice Dataguide**

Formalmente, um caminho  $p$  é uma sequência de vértices  $v \in G'$  ligados a partir do vértice *RAIZ*. Para um dado caminho  $p$ , visto como uma cadeia de vértices, temos que o último vértice da cadeia nunca aparece como último vértice de qualquer outro caminho. Portanto, cada caminho  $p$  é associado ao último vértice do caminho  $p$  de forma que: *o conjunto  $S$  de identificadores de documentos associado a  $v \in G'$ , que é o último vértice do caminho  $p$  a partir do vértice *RAIZ*, indica o conjunto de documentos no qual ocorre o caminho  $p$  na fonte de dados.*

Vale ressaltar que no trabalho original de Goldman [Goldman97] não são considerados valores de atributos na geração do Dataguide, isto é, somente a parte estrutural é indexada. O Dataguide é um índice navegacional sob a estrutura dos documentos que fica em memória principal, de forma que os valores de atributos são armazenados em memória secundária. Desta forma, para encontrar os documentos que contém um determinado caminho com um determinado valor, inicialmente a parte estrutural é consultada em memória e, após isso, os valores de dados relacionados ao caminho encontrado na primeira etapa são carregados a partir da memória secundária para uma segunda etapa de busca. Nesta segunda etapa, são selecionados somente os caminhos com valores que correspondem ao expressado na consulta.

Apesar de o Dataguide não ter sido diretamente projetado para suportar o modelo XML, a técnica pode ser aplicada em qualquer modelo de dados baseado em grafos [Goldman97]. Neste sentido, para possibilitar uma comparação do BranchGuide (a técnica proposta nesta dissertação) em relação ao Dataguide, é adotada uma adaptação do Dataguide para o modelo de representação de dados XML. Tal adaptação implica que ao computar o grafo de índice, os

valores de atributos também são considerados. Sendo assim, no caso de vértices de atributos, estes somente são mesclados se tiverem o mesmo nome de atributo e mesmo valor associado. Na Figura 5, o último vértice do caminho indexado *editora-livro-autor-nome* – “Shakespeare” é um exemplo de vértice de atributo mesclado, já que ocorre simultaneamente nos Documentos 1 e 2.

### 2.1.2 O Problema do Processamento de Consultas com Ramificação

O Dataguide é classificado como um *índice baseado em caminhos simples*, já que processa sem perda de precisão somente *expressões de caminhos simples*. Estas expressões representam uma cadeia linear de navegação sob o índice. Expressões de caminhos simples são convenientes para processar somente consultas com única condição de restrição apenas no final da expressão. Com o Dataguide não é possível processar *expressões de ramificação*, as quais representam sequências de navegação que contêm ramificações (bifurcações), correspondendo a uma pequena árvore, onde para cada ramificação existe uma condição de restrição. Resumidamente, este problema advém porque quando indexamos uma fonte de dados com o Dataguide ocorre perda de informações. Este problema é detalhado a seguir.

Por exemplo, a Figura 6a mostra um documento descrevendo os livros publicados pela editora *Simon&Schuster*, que publica os livros *Hamlet* escrito por *Shakespeare* e *Frankenstein* escrito por *Mary Shelley*. Através da especificação de expressões de consultas, usuários podem procurar por editoras que publicam livros específicos. Suponha que o usuário tenha a seguinte (mas bizarra) necessidade de informação: “*encontrar editoras que publicam o livro Frankenstein escrito por Shakespeare*”. Esta necessidade de informação é mapeada para a expressão de ramificação ilustrada na Figura 6b.

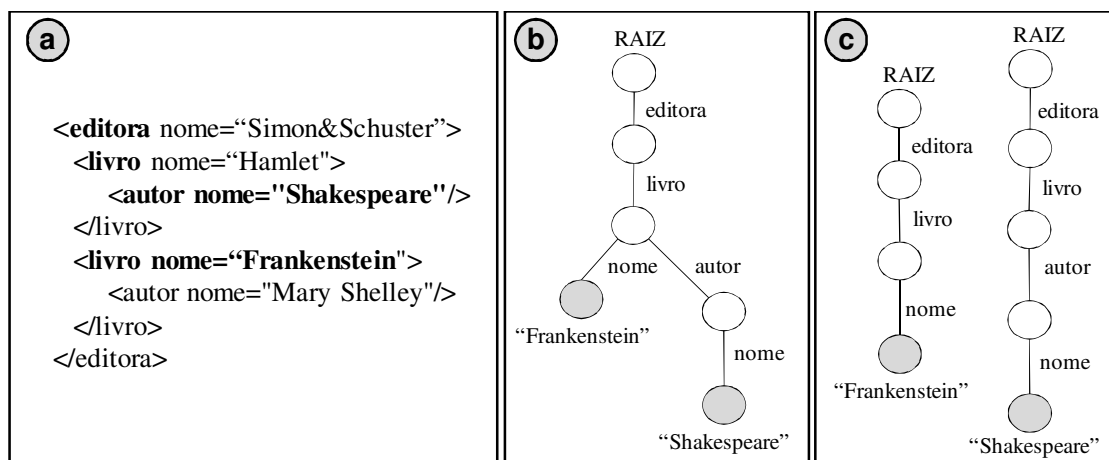


Figura 6: Expressões com ramificação e decomposição em expressões simples

A fim de tratar a expressão de consulta com ramificação (é importante notar a bifurcação na representação da consulta, caracterizando uma pequena árvore) ilustrada na Figura 6b; qualquer técnica baseada em caminhos simples (tal como o Dataguide) deve decompor a árvore em duas expressões de caminhos simples, ilustradas na Figura 6c. Então, estas expressões de caminho simples podem ser empregadas individualmente para acessar o índice e retornar os documentos que contém a ocorrência dos dois caminhos simultaneamente. Temos que no documento mostrado na Figura 6a ocorrem os dois caminhos (destacados em negrito na Figura 6a), e assim, o documento é retornado como um resultado. Entretanto, claramente, o documento **não** satisfaz a necessidade de informação do usuário. Este falso positivo ocorre porque a relação entre o livro e seu autor não é mantida após indexação pela técnica de indexação Dataguide.

Uma maneira de visualizar a perda de informações após a indexação é comparando o grafo de dados e o respectivo grafo de índice gerado pelo Dataguide, ilustrados respectivamente na Figura 7a e Figura 7b. Temos que a execução de consultas sob o grafo de índice pode ser vista como um problema de sobreposição de grafos: *se o grafo da consulta com ramificação cobrir o grafo de índice, então existem documentos que satisfazem as condições impostas na consulta*. Logo, o falso positivo ocorre porque no grafo de índice (Figura 7b) existe uma cobertura em relação à expressão com ramificação da Figura 6b, mas em compensação, no grafo de dados (Figura 7a), esta cobertura não ocorre, indicando que o documento indexado não satisfaz a restrição de consulta, e desta forma, temos a ocorrência de um falso positivo.

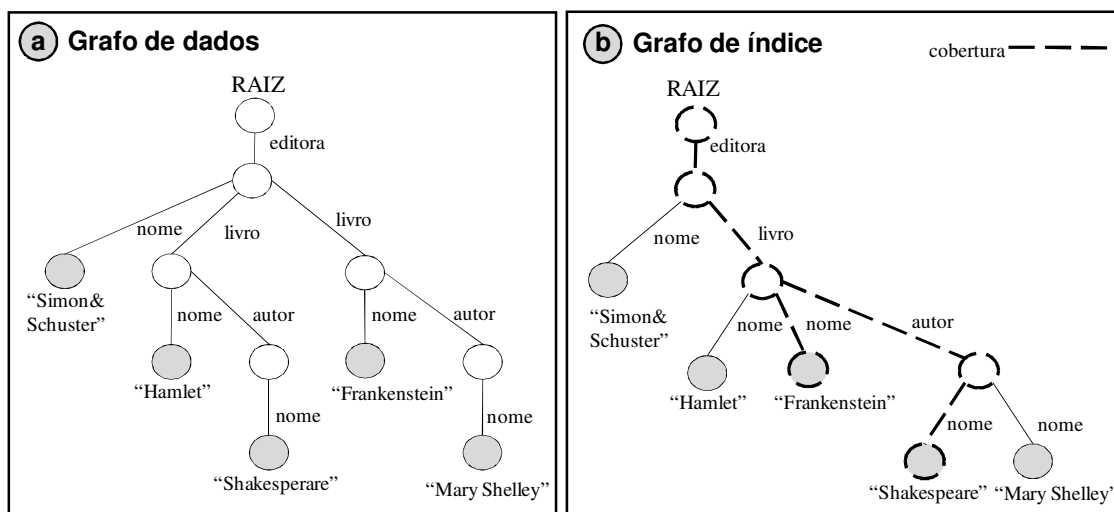


Figura 7: Perda de informações após indexação

## 2.2 Índices de Interseção Estrutural

Como apontado por Milo [Milo99], um índice mapeia um determinado domínio (por exemplo, um conjunto de palavras-chave, caminhos de navegação) para um conjunto de ocorrências (por exemplo, identificadores de documentos, identificadores de elementos) cuja definição varia de acordo com o índice. Em outras palavras, temos que todo índice possui uma interface que especifica o tipo de objeto de entrada (a consulta) e o tipo de objeto de saída (o resultado obtido após a pesquisa).

Desta forma, no caso do Dataguide, como discutido na Seção 2.1.1, a interface deste índice aceita como entrada expressões de caminhos simples e o resultado obtido após pesquisa são identificadores de documentos que contêm o caminho simples informado. Caminhos com ramificação podem ser decompostos em expressões de caminho simples e uma interseção pode ser feita com os resultados intermediários, mas, como demonstrado, isso implica na possibilidade de retorno de falsos positivos.

Em contraste, outra categoria de índices de documentos XML não sofre deste problema devido à utilização de diferentes interfaces de índice e métodos de processamento de resultados intermediários que são detalhados a seguir.

As técnicas de indexação denominadas *índices de interseção estrutural* não têm como entrada expressões de caminhos extraídos de uma fonte de dados. Especificamente, índices de interseção estrutural são compostos de dois índices: o primeiro mapeia nomes de elementos XML para identificadores de elementos; e o segundo mapeia nomes de atributos XML juntamente com respectivos valores associados para identificadores de atributos. Estas ideias foram inicialmente apresentadas no sistema XISS (*XML Indexing & Storage System*) [Li01].

O identificador de um elemento  $e$  é determinado por um par  $(d, i_e)$  em que  $d$  é o identificador global (isto é, em relação a todos os documentos de uma determinada fonte de dados) do documento em que ocorre o elemento XML  $e$ , além disso,  $i_e$  é um identificador local do elemento XML em relação ao documento  $d$ . Já o identificador de um atributo  $a$  é determinado por um par  $(d, (i_a, t))$ , em que  $d$  é o identificador global do documento no qual ocorre o atributo XML  $a$ , e  $(i_a, t)$  é um identificador local do atributo XML em relação específica ao documento em que ocorre  $a$  com o valor de atributo  $t$  associado. Em geral, índices que mapeiam os nomes de atributos e elementos para os identificadores citados provêm o suporte mínimo necessário para o processamento de consultas com técnicas baseadas em *interseção estrutural*.

No processamento de consultas, as técnicas de indexação baseadas em interseção estrutural empregam, em uma visão abstrata, um tipo de algoritmo em que a consulta é decomposta em vários termos. Cada termo corresponde a um nome de elemento ou atributo XML. Com isto, uma consulta ao índice de elemento ou atributo é realizada para cada termo que compõe a consulta inicial a fim de obter resultados intermediários associados. Tais resultados intermediários são conjuntos contendo identificadores de elementos ou atributos.

Por exemplo, dado um caminho “ $a-b$ ” no qual  $a$  e  $b$  são elementos XML, o índice de elementos é acessado duas vezes e é retornado um conjunto de identificadores de elementos  $S_a = \{(d,i) \therefore \text{elemento } a \text{ ocorre no endereço } i \text{ do documento } d\}$  e  $S_b = \{(d,i) \therefore \text{elemento } b \text{ ocorre no endereço } i \text{ do documento } d\}$ . Estes conjuntos correspondem respectivamente ao conjunto de ocorrências dos elementos de nome  $a$  e  $b$  existentes na base de dados.

Após isso, individualmente, para cada identificador de documento  $d$ , o processador de consultas combina os conjuntos  $S_a$  e  $S_b$  através de um algoritmo de *interseção estrutural* a fim de obter o resultado final da execução da consulta. Este algoritmo permite checar se ocorrem as relações de descendência entre elementos definidas na consulta “ $a-b$ ”. Isto é, são retornados os elementos  $b$  tal que estes sejam filhos do elemento  $a$ . Este procedimento de interseção estrutural será detalhado a seguir.

### 2.2.1 eXist

O *eXist* [Meier02] é um sistema *open-source* para armazenamento de documentos XML que emprega técnicas de interseção estrutural similares aos inicialmente propostos em [Li01]. A principal diferença em relação ao *XISS* [Li01] é que ao invés de utilizar o esquema de identificação proposto originalmente em [Li01], é adotado um esquema de identificação baseado no sistema de classificação decimal de *Dewey* [Timo04], em que os identificadores são compostos por uma sequência de valores numéricos separados por pontos.

Um exemplo de conjunto de índices do *eXist* gerado a partir do documento  $D$  (mostrado na Figura 8a) é utilizado para fins de discussão e exemplificação. O documento  $D$  é disposto em forma hierárquica de forma que para cada elemento ou atributo é atribuído um identificador baseado no sistema de classificação decimal *Dewey* de acordo com o sentido de navegação *pai*→*filho* a partir do elemento *editora*.

Por exemplo, o elemento *editora* é identificado por 1. O elemento *editora* possui o elemento filho *livro*, cujo identificador é 1.2. Por fim, o elemento *livro* possui o elemento filho *autor*, numerado por 1.2.2. É importante notar que com este esquema de numeração,

dado dois identificadores de elemento ou atributo podem-se inferir as relações de descendência entre os mesmos. Para isto basta uma comparação de prefixos dos identificadores. Por fim, todo identificador tem o documento  $D$  agregado, de forma que, os elementos e atributos de um documento  $D$  não podem ter relações de descendência com elementos de outro documento  $D' \neq D$ .

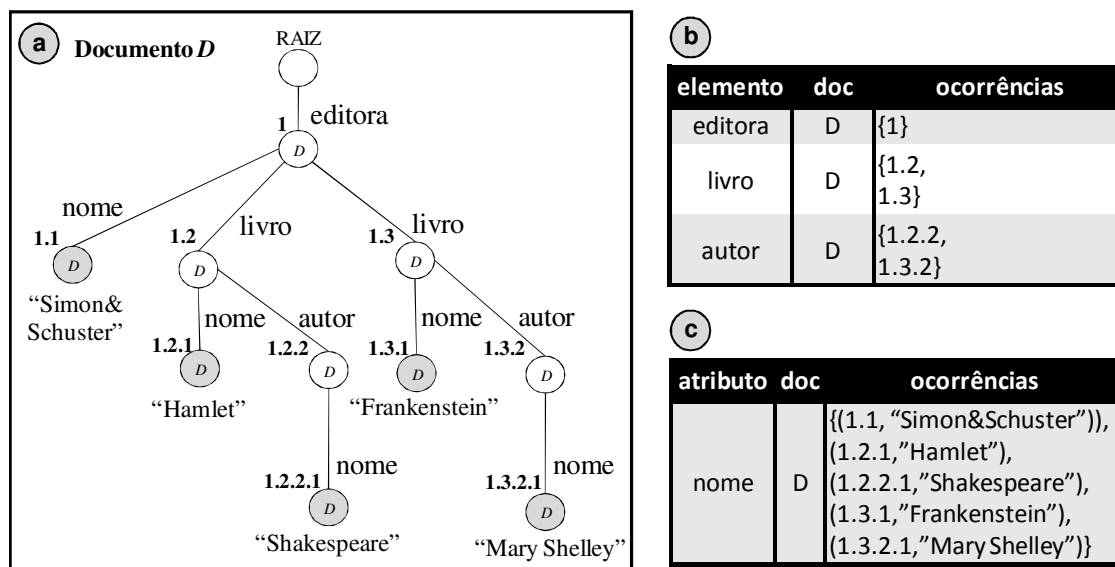


**Figura 8: Exemplo de índice gerado pelo eXist**

No processamento de uma consulta relacionada a expressões de caminhos simples ou com ramificação, o *eXist* decompõe a expressão em termos individuais. De posse destes termos, é executado um acesso ao índice, bem como uma etapa parcial de interseção de conjuntos no tempo de execução da consulta, até chegar a um conjunto de resultado final. Para fins de exemplificação, a seguir, é detalhado o processamento da consulta com ramificação ilustrada na Figura 6b, mas desta vez, utilizando índices baseados em interseção estrutural.

Inicialmente todos os nomes existentes na consulta são decompostos, de maneira que obtemos o conjunto de termos  $T = \{editora, livro, nome \text{ com valor "Frankenstein"}, autor, nome \text{ com valor "Shakespeare"}\}$ . De posse destes termos e de acordo com a hierarquia entre elementos e atributos definida na expressão com ramificação, temos que o processamento se dá de acordo com os seguintes passos (ilustrados sequencialmente nas Figura 9a, Figura 9b, Figura 9c, Figura 9d e Figura 9e):

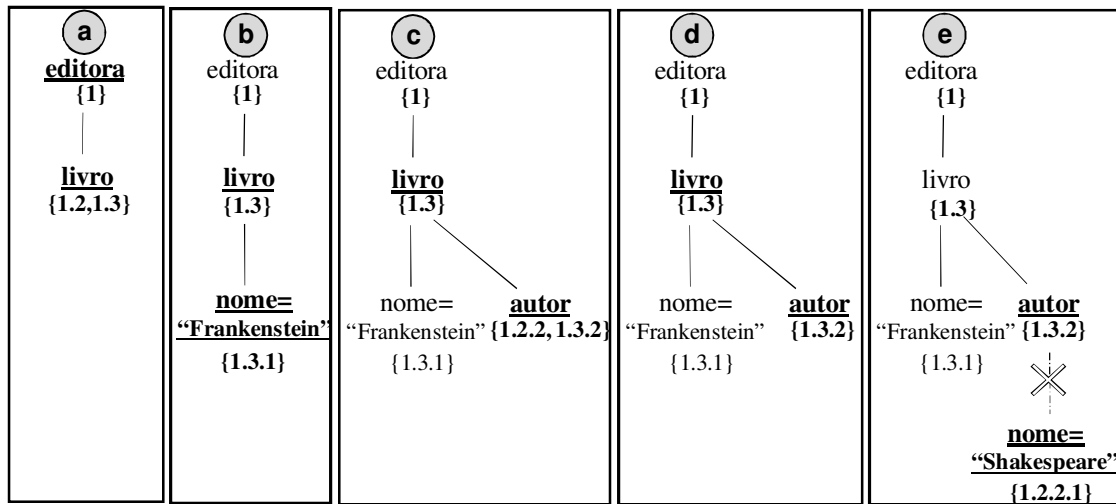


**Figura 9: Processamento de consulta utilizando interseção estrutural**

a) Primeiramente, a partir do índice de elementos (Figura 8b) são recuperados os dois conjuntos,  $S_{editora}$  e  $S_{livro}$ , contendo identificadores de elementos denominados *editora* e *livro*, respectivamente. Na interseção estrutural (Figura 9a), é mantido no conjunto  $S_{livro}$  todos identificadores com o prefixo igual ao valor do identificador do único elemento do conjunto  $S_{editora}$ .

b) Em um segundo passo, a partir do índice de atributos (Figura 8c) é recuperado o conjunto  $S_{nome="Frankenstein"}$  contendo identificadores de atributo denominados *nome* tal que seu valor seja “Frankenstein”. Na interseção estrutural são mantidos no conjunto  $S_{nome="Frankenstein"}$  somente os identificadores de atributos com o prefixo igual ao valor de identificador de algum elemento do conjunto  $S_{livro}$ . Além disso, no conjunto  $S_{livro}$  são mantidos somente os identificadores no qual existe algum identificador em  $S_{nome="Frankenstein"}$  que tenha como prefixo algum identificador contido no conjunto  $S_{livro}$ . Sendo assim, como o identificador 1.3.1 que pertence ao conjunto  $S_{nome="Frankenstein"}$  tem o prefixo 1.3 contido no conjunto  $S_{livro}$ , então o identificador 1.3.1 é mantido no conjunto (Figura 9b). Ao contrário, como no conjunto  $S_{nome="Frankenstein"}$  não existe nenhum elemento com prefixo 1.2, então o identificador 1.2 é removido do conjunto  $S_{livro}$  (Figura 9b). De uma maneira mais simplificada, estas duas checagens garantem que nos conjuntos relativos ao elemento livro e do atributo nome, somente restarão identificadores em que exista uma relação de descendência *pai*→*filho* do tipo *livro*-*nome* (tal que seu valor seja “Frankenstein”) preservada para pós-processamento.

c) O conjunto de identificadores  $S_{autor}$  de elementos de nome *autor* é recuperado (Figura 9c) a partir do índice de elementos (Figura 8b).

d) Analogamente ao caso ilustrado na Figura 9b, por não satisfazer as relações de descendência  $pai \rightarrow filho$ , na interseção estrutural é filtrada uma das ocorrências (especificamente o identificador 1.2.2) do conjunto do elemento  $S_{autor}$  (Figura 9d).

e) Por fim, é recuperado o conjunto  $S_{nome="Shakespeare"}$  de ocorrências de atributos *nome* com valor “Shakespeare” (Figura 9e). Neste caso é retornada uma ocorrência, mas esta não satisfaz a relação de descendência  $pai \rightarrow filho$  com o elemento *autor*, gerando assim um conjunto vazio  $\{\emptyset\}$  na interseção estrutural. Com isto, não existe nenhum documento que satisfaça a consulta com ramificação do exemplo, isto é, o documento  $D$  (Figura 8a) não satisfaz a consulta com ramificação mostrada na Figura 6b.

Como mostrado neste exemplo, a técnica de indexação adotada no eXist emprega interseção estrutural e não retorna falsos positivos na execução de expressões com ramificação, ao contrário da técnica Dataguide apresentada na Seção 2.1.1. Isto ocorre porque no processamento de consultas cada identificador de elemento/atributo recuperado preserva sem perda de precisão as relações estruturais do documento, assim sendo, não existe a possibilidade de retorno de falsos positivos, tanto na execução de expressões de caminho simples, bem como de expressões de caminho com ramificação. Apesar desta vantagem, como efeito colateral, as interseções estruturais implicam em um maior custo de processamento, o que será discutido na próxima subseção.

### 2.2.2 O Problema do Custo de Interseções

Índices baseados em caminhos indexam um subconjunto definido de caminhos da fonte de dados (o que causa perda de precisão); já nos índices baseados em interseção estrutural, ao invés dos caminhos, são indexados elementos e atributos. Com isto é possível checar relações expressas através de consultas com ramificação, pois estas são decompostas em objetos de menor granularidade (elementos/atributos) que estão diretamente indexados sem qualquer perda de informação. Apesar disso, a adoção de esquemas de indexação direta de elementos/atributos conduzem para um alto custo de armazenamento [Catania05].

Em um cenário comum, devemos ter mais do que um documento na fonte de dados quando a premissa é localizar documentos dentre uma base de documentos que satisfaçam restrições definidas. Uma base de dados maior implica em uma ocorrência maior de atributos e elementos indexados. Sendo assim, quanto maior a quantidade de documentos maior é a quantidade de etapas de interseção geralmente executadas, já que mais ocorrências de

atributos/elementos indexados devem ser checadas e possivelmente filtradas durante o processamento de consultas.

Por fim, vale ressaltar que as interseções estruturais são calculadas em tempo de execução e por isso podem degradar o desempenho de execução das consultas. Operações de interseção estrutural têm alto custo de processamento, e, como demonstrado experimentalmente em [Kaushik04], a melhoria de desempenho no processamento de consultas depende da redução da quantidade de operações de interseção estrutural.

## 2.3 Índices de Sequência

Visto que a melhoria de desempenho no processamento de consultas depende da redução da quantidade de operações de interseção estrutural executadas, então, abordagens em que não é necessário qualquer tipo de interseção estrutural têm sido investigadas (tais como [Wang03], [Wang05]). Mais especificamente, os *índices de sequência* adotam uma representação que reflete o conteúdo completo do documento, de forma que esta representação é a unidade básica de indexação, e assim, não existe decomposição de consultas (em elementos ou atributos individuais) na pesquisa aos documentos semi-estruturados. Neste sentido, na seguinte subseção, um índice de sequência *ViST* [Wang03] é apresentado e discutido.

### 2.3.1 ViST

No índice de sequência *ViST* (*Virtual Suffix Tree*) [Wang03] a geração da forma de representação de um documento segue o procedimento adiante: para cada documento  $D$ , tal que  $G = \text{grafo de dados de } D$ , é gerada uma sequência  $s$  a partir de um percurso em pré-ordem sob  $G$  (recursivamente visita-se cada vértice do grafo  $G$  da esquerda para a direita). Cada item da sequência  $s$  é composto pelo nome de uma aresta de elemento ou o nome de uma aresta de atributo com o seu respectivo valor de atributo associado.

Por exemplo, os documentos  $D_1$  e  $D_2$  (ambos mostrados na Figura 2) geram, respectivamente, os grafos de dados mostrados na Figura 10a e Figura 10b. O valor das sequências  $s_1$  e  $s_2$  respectivas à visitação em pré-ordem sob os grafos dos documentos  $D_1$  e  $D_2$  são mostradas na Figura 10c e Figura 10d, respectivamente. Vale ressaltar que nas figuras Figura 10a e Figura 10b, os valores ordinais no canto superior esquerdo de cada vértice indicam a correlação ordinal em que os vértices são visitados.

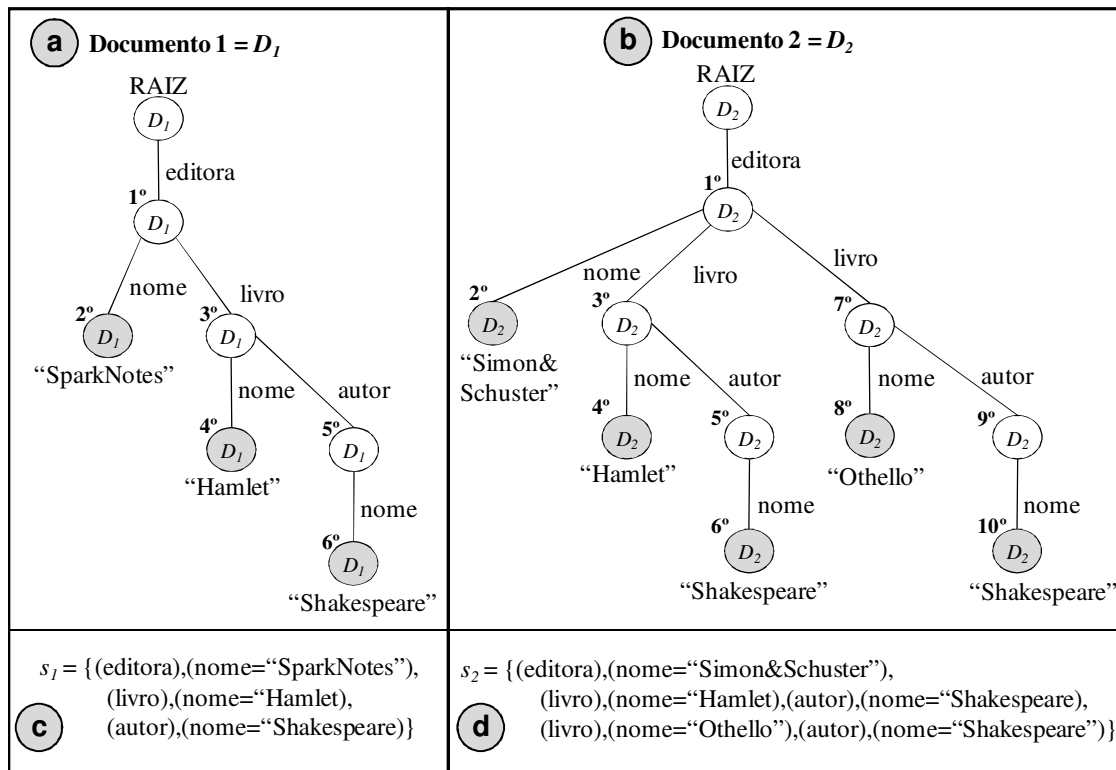


**Figura 10: Sequências  $s_1$  e  $s_2$  geradas a partir dos documentos  $d_1$  e  $d_2$**

No índice ViST, pesquisar em uma fonte de dados de documentos XML indexada é equivalente a pesquisar subsequências não contínuas dentre o conjunto total de sequências geradas a partir de todos os documentos da fonte de dados. Logo, dada uma consulta (que é vista como um pequeno grafo), esta também é transformada em uma sequência adotando o mesmo procedimento usado na indexação dos documentos. A comparação de subsequências não contínuas é utilizada no processamento de consultas para recuperar documentos.

O benefício de transformar o processo de execução de consultas em um processo de comparação de sequências é que as consultas são processadas por inteiro ao invés de serem decompostas em unidades menores (tais como conjuntos de caminhos, elementos ou atributos). Neste sentido, já que não são combinados resultados parciais através de operações de intersecção estrutural, o custo de processamento é reduzido [Wang03].

### 2.3.2 O Problema de Filtragem de Falsos Positivos

A estrutura de dados amplamente adotada para armazenamento e busca de sequências é denominada *árvore de sufixos* [Wang03]. Uma árvore de sufixos armazena de forma compacta um conjunto de sequências através da não replicação dos sufixos do conjunto de sequências. A busca de uma sequência inicia a partir do nó raiz da árvore de sufixo até ser encontrado o último nó correspondente da sequência procurada. Os nós das árvores de sufixo

podem armazenar informações auxiliares, como por exemplo, a listagem de documentos em que ocorre uma dada sequência. Assim é possível recuperar a lista de documentos em que ocorre uma determinada sequência.

Por exemplo, tomando as sequências  $s_1$  e  $s_2$  ilustradas na Figura 10c e Figura 10d respectivamente, temos que após inserção destas sequências em uma árvore de sufixos  $T_{sf}$  é obtido como resultado final a árvore de sufixos  $T_{sf}$  ilustrada na Figura 11.

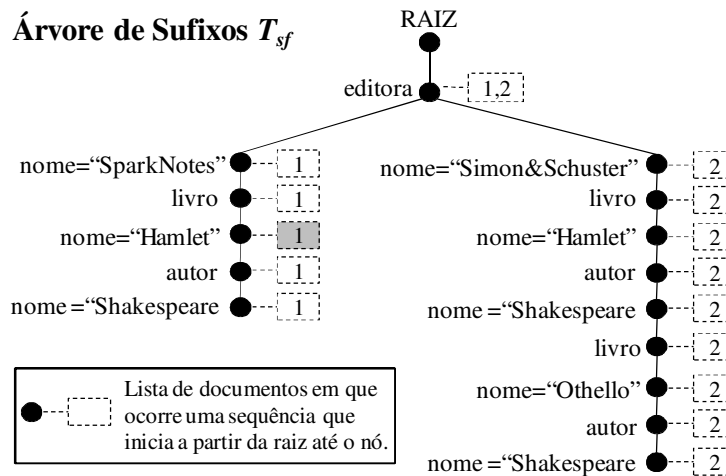


**Figura 11: Árvore de sufixos  $T_{sf}$  após inserção das sequências  $s_1$  e  $s_2$  da Figura 10**

Perceba que como o único prefixo coincidente das sequências  $s_1$  e  $s_2$  é o item (*editora*), assim temos que o nó filho da raiz *RAIZ* da árvore  $T_{sf}$ , rotulado como *editora* é armazenado uma única vez. Além disso, é indicado na lista de informações associadas (especificamente, o conjunto numérico  $\{1, 2\}$ ) ao respectivo nó que a sequência ocorre nos documentos  $D_1$  e  $D_2$ . Apesar disso, as sequências  $s_1$  e  $s_2$  não têm o segundo item de sequência igual, desta forma, gerando uma bifurcação na árvore de sufixos  $T_{sf}$ .

O objetivo do uso da árvore de sufixo é procurar por sequências indexadas e assim recuperar documentos que contêm a ocorrência de uma dada sequência. Por exemplo, dada a sequência  $s_q = \{(editora), (nome = "SparkNotes"), (livro), (nome = "Hamlet")\}$ , percorrendo  $T_{sf}$  desde o nó *RAIZ* de acordo com os itens definidos em  $s_q$  temos o seguinte percurso  $P_r$  sob os nós de  $T_{sf}$ :  $P_r = RAIZ \rightarrow (editora) \rightarrow (nome = "SparkNotes") \rightarrow (livro) \rightarrow (nome = "Hamlet")$ . Assim, é obtida a confirmação positiva de que a sequência  $s_q$  está contida em  $T_{sf}$ .

O último nó visitado no percurso  $P_r$  indica a lista de documentos em que existe a ocorrência da sequência procurada. Em  $P_r$ , o último nó corresponde ao item (*nome = "Hamlet"*) e as informações associadas a este nó (destacadas na Figura 11 através de sombreamento em cinza) indicam que a sequência  $s_q$  somente ocorre em  $D_1$ .

O emprego da árvore de sufixos reduz o espaço de armazenamento, pois diminui a replicação de nós relativos aos prefixos coincidentes entre o conjunto de sequências geradas a partir de todos os documentos da fonte de dados. Semelhante ao índice Dataguide, a redução de informações após processo de indexação causa perda de precisão. Vale ressaltar que, no pior caso, para uma base de dados em que todos os documentos são diferentes, de forma que não haja coincidência de prefixo, temos que o tamanho da árvore de sufixos é linear à fonte de dados. Sendo assim, o ViST não gera tamanhos de índices maiores do que o tamanho da fonte de dados indexada.

Por um lado, o ViST possui a propriedade de gerar tamanhos de índices geralmente menores do que o tamanho da fonte de dados, por outro lado, o ViST não mantém todas as relações existentes entre elementos e atributos dos documentos. Dentre tais relações destacam-se as necessárias para processar consultas com ramificação. Devido à imprecisão do índice, no ViST, o processamento de consultas com ramificação pode retornar falsos positivos. Para tratar esse problema, em geral, técnicas de filtragem são aplicadas ao resultado de pesquisa obtido a fim de eliminar os falsos positivos [Catania05]. Estas técnicas geralmente são: o processamento direto de documentos após recuperação via índice ou utilização de operações de interseção estrutural adicionais. Ambos os recursos implicam em um custo adicional no tempo de processamento de consultas [Wang05].

Para uma melhor compreensão do leitor, a seguir, é mostrado um caso em que ocorrem falsos positivos ao adotar a técnica de indexação ViST. Além disso, são destacadas as causas da geração de falsos positivos. Neste sentido, a seguir é revisitado o caso discutido nas Seções 2.1 e 2.2, porém, desta vez, empregando a técnica de indexação ViST.

Na Figura 12a e Figura 12b o documento  $D$  (Figura 8a) é ilustrado de duas formas diferentes: *i)* Na Figura 12a, o documento  $D$  é mostrado em uma primeira forma, que de acordo com o percurso em pré-ordem sob  $D$  gera a sequência  $S_D$  também mostrada na Figura 12a. *ii)* Já na Figura 12b, a ordem no qual os dois elementos irmãos *livro* (que são filhos do mesmo elemento pai *editora*) é alterada no momento da geração da sequência através de uma simples permuta sob qual sub-árvore deve ser visitada primeiramente no percurso em pré-ordem. É importante ressaltar que esta permuta não afeta as propriedades semânticas do documento  $D$ , de forma que o documento continua armazenando as mesmas informações tanto em termos estruturais quanto em termos de valores textuais descritos. O percurso em pré-ordem sob  $D$  adotado no segundo caso gera a sequência  $S'_D$  (ilustrada na Figura 12b).

Ambas as sequências  $S_D$  e  $S'_D$  representam o documento  $D$  e podem ser adotadas para indexação.

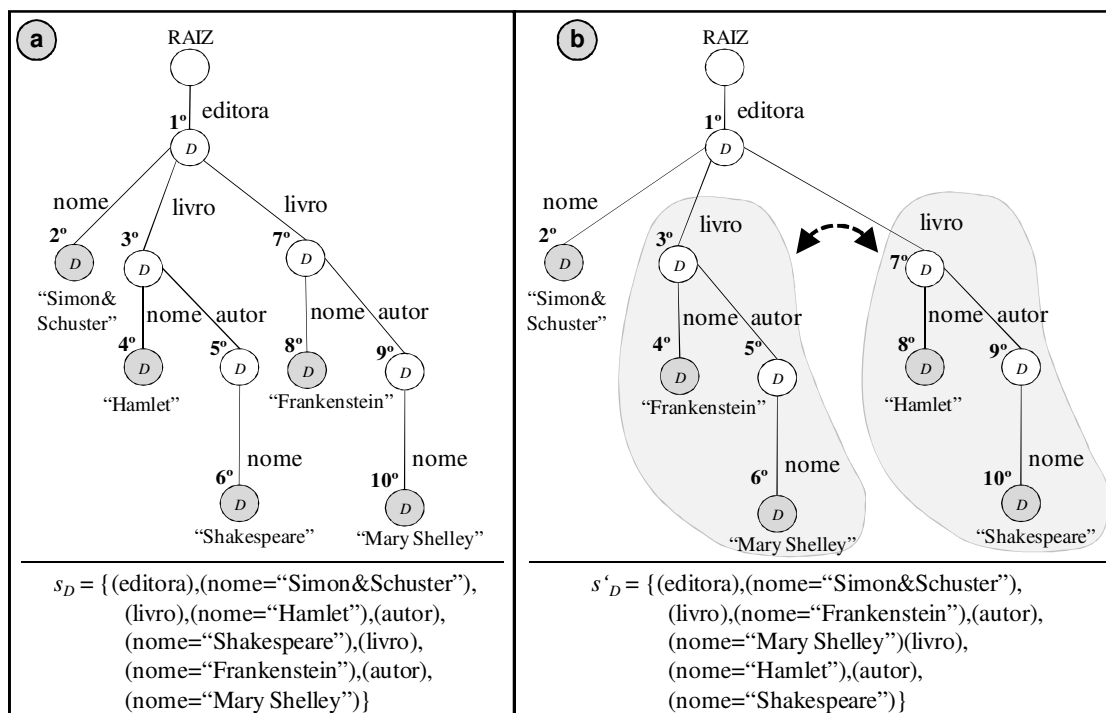


**Figura 12: Alteração de ordem de elementos na geração de sequência**

No caso em que é pesquisado pela seguinte informação bizarra: “*editora que publique o livro Frankenstein escrito por Shakespeare*”; Obviamente, o ViST **não** deve retornar nenhum resultado para esta consulta. A estrutura da consulta e sequência gerada após navegação em pré-ordem é ilustrada na Figura 13. Neste contexto, temos que  $S_q$  é a sequência gerada a partir da consulta e deve ser comparada com as sequências  $S_D$  e  $S'_D$  com o objetivo de verificar se o documento  $D$  satisfaz as restrições especificadas.

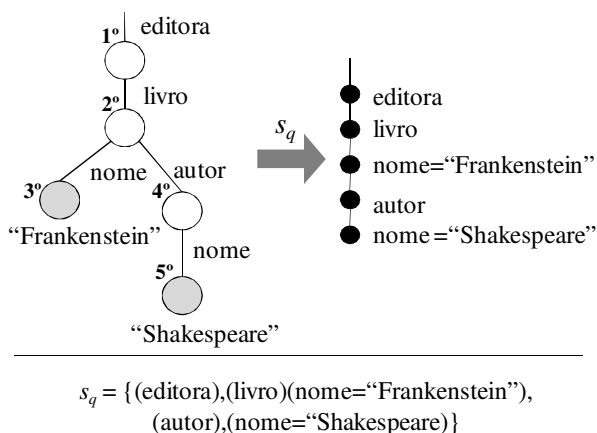


**Figura 13: Geração da sequência  $S_q$  a partir de informações da consulta**

Formalmente, uma sequência  $S$  satisfaz as restrições de consulta representadas pela sequência  $S_q$  se as seguintes condições forem verdadeiras:

- Sendo  $S_q = \{(q_1, q_2, \dots, q_n)\}$  e  $S = \{(r_1, r_2, \dots, r_m)\}$  com  $m \geq n$ . Para todo item  $q_i \in S_q$  existe um item  $r_i \in S$  tal que  $q_i = r_i$ .
- Além disso, para todo par de elementos  $(q_i, q_j) \in S_q$  tal que  $j > i$ , temos que o item  $q_j$  deve ser encontrado após o item  $q_i$  no percurso sob  $S$ .

O procedimento de verificação pode ser descrito resumidamente da seguinte forma: Se para todo item de  $S_q$  for sequencialmente encontrada, mas não necessariamente de forma sucessiva, uma ocorrência de item de sequência igual em  $S$ , então, o documento representado pela sequência  $S$  satisfaz as restrições de consulta representadas pela sequência  $S_q$ .

Na Figura 14 é ilustrado o efeito da verificação sob o exemplo discutido nesta seção. Na Figura 14a e Figura 14b uma seta apontando de  $S_q$  para  $S_D / S'_D$  (respectivamente) denota que um item da sequência de consulta tem, de acordo com o procedimento de verificação, um item correspondente satisfeito. Vale ressaltar que na Figura 14a não existe item correspondente para o item (*nome*="Shakespeare"), de forma que o documento  $D$  (representado pela sequência  $S_D$ ) não é retornando como resultado da consulta, e este é o resultado **correto** e esperado. Em contrapartida, na Figura 14b, existem itens correspondentes em  $S'_D$  para todos os itens da sequência  $S_q$ , de forma que o documento  $D$  é **retornado**, mas é um **falso positivo**.

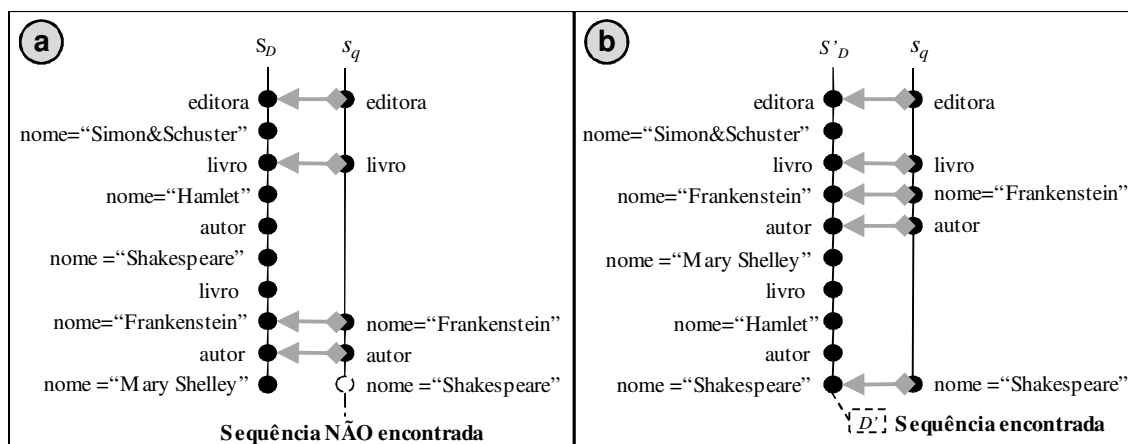


**Figura 14: Comparação da sequência  $S_q$  com as sequências  $S_D$  e  $S'_D$**

Como apontado por [Wang05], o problema de retorno de falsos positivos é existente entre a maioria dos métodos de indexação baseado em sequências. Especificamente, a ocorrência de elementos XML irmãos que tem o mesmo nome gera este problema na representação de dados a ser indexada, pois dependendo da quantidade de elementos irmãos

de mesmo nome podemos ter um maior conjunto de diferentes formas de atribuição de ordem a cada elemento irmão, e assim, gerar diferentes resultados de ordenação dos elementos. Cada diferente ordenação gera uma diferente representação em sequência para o mesmo documento. No exemplo apresentado na Figura 12, foram geradas duas diferentes representações devido à ocorrência dos elementos irmão *livro* de mesmo nome.

Desta forma, pode ocorrer um conjunto ambíguo de consultas associado a um conjunto de documentos [Wang05], ou seja, consultas podem ser associadas a resultados que não correspondem ao que realmente foi especificado nas mesmas, e daí, gerando falsos positivos.

## 2.4 Considerações Finais

Nesta seção são apresentados resumidamente os principais pontos de discussão deste capítulo ao mesmo tempo em que são antecipadas algumas características da técnica de indexação BranchGuide. Especificamente, os trabalhos Dataguide [Goldman97], eXist [Meier02] e ViST [Wang03] são comparados com a técnica BranchGuide em função dos quesitos listados na Tabela 1. Apesar de algumas propriedades da técnica BranchGuide serem brevemente citadas nesta seção, é importante ressaltar que tais propriedades são detalhadas e discutidas nos Capítulos 3 e 4.

**Tabela 1: Tabela comparativa**

|                                                         | Dataguide                                      | eXist                                      | ViST                                          | BranchGuide                                         |
|---------------------------------------------------------|------------------------------------------------|--------------------------------------------|-----------------------------------------------|-----------------------------------------------------|
| Expressões Suportadas:                                  | Expressões simples                             | Expressões com ramificação                 | Expressões com ramificação                    | Classe de expressões com ramificação                |
| Precisão de Resultados para Expressões com Ramificação: | Não provê suporte a expressões com ramificação | Sim                                        | Não                                           | Sim                                                 |
| Método Empregado no Processamento de Expressões:        | Busca Navegacional                             | Interseção Estrutural                      | Árvore de Sufixos                             | Índice Invertido                                    |
| Custo para o Armazenamento de Índice:                   | Igual ou menor ao tamanho da amostra indexada  | Maior do que o tamanho da amostra indexada | Igual ou menor ao tamanho da amostra indexada | Igual ou maior do que o tamanho da amostra indexada |

O índice Dataguide é conveniente para processar somente consultas com única condição de restrição apenas no final da expressão (expressões simples). Apesar disto, o procedimento de decomposição de expressões com ramificação em expressões simples com o

Dataguide não garante que os resultados finais obtidos no processamento de expressões com ramificação sejam precisos.

Já o eXist emprega intersecção estrutural, assim, consultas com ramificação são decompostas em termos individuais, de forma que cada termo individual tem um identificador unívoco que permite checar diversas relações entre elementos, e desta forma consultas podem ser processadas sem retorno de falsos positivos. A técnica ViST é projetada para processar qualquer tipo de consulta com ramificação mas os resultados retornados pela mesma podem incluir falsos positivos. Já a técnica BranchGuide proposta neste trabalho adota um padrão para a seleção de uma classe de expressões com ramificação a fim de que um conjunto de expressões com ramificação seja indexado sem imprecisão. Além disso, como também indexa caminhos simples, a técnica BranchGuide provê suporte para consultas mais expressivas do que o Dataguide. Vale ressaltar também que, o ViST retorna falsos positivos ao processar consultas com ramificação que pertencem à classe de consultas de ramificação suportada pela técnica BranchGuide.

As técnicas de indexação Dataguide e ViST geram após indexação, no pior caso, uma quantidade linear de informações a serem indexadas [Goldman97] [Wang03], ou seja, em geral, os índices gerados têm tamanho menor ou igual ao tamanho da amostra indexada. Em contraste, o eXist sofre de problemas para prover escalabilidade. Especificamente em relação aos custos de armazenamento de índice, o fato do eXist adotar um esquema de identificação baseado em numeração Dewey conduz para um alto custo de armazenamento [Catania05], que em geral, causa um elevado tamanho de índice em comparação com o tamanho da amostra indexada. Em contraste, embora a técnica BranchGuide gere uma quantidade não linear de expressões com ramificação indexadas, é adotado um projeto de índice que reduz o crescimento dos tamanhos de índice, provendo requisitos de armazenamento menos custosos do que o eXist. Como evidenciado experimentalmente (Capítulo 6), os tamanhos de índice gerados pela técnica BranchGuide são em geral, próximos (podendo ser iguais) ou maiores do que o tamanho da amostra indexada, mas em compensação, são menores do que os índices gerados pelo eXist.

Em relação aos métodos empregados no processamento de consultas, temos que o eXist decompõe consultas com ramificação em termos individuais, assim, um acesso ao índice e uma etapa de intersecção estrutural deve ser executado para cada um destes. No total, como demonstrado experimentalmente em [Kaushik04], a soma dos tempos de acesso e

interseções estruturais implica, em geral, em um método de alto custo para o processamento de consultas.

Já o Dataguide adota uma estrutura de dados de busca navegacional em memória [Goldman97], especificamente, um grafo mantido em memória principal tal que é possível percorrê-lo direcionalmente. Este grafo contém todos os caminhos simples, de forma que o processo de consulta se resume a localizar nessa estrutura de dados um percurso relativo a um determinado caminho (navegando sob os vértices do grafo em memória), sem a necessidade de decomposição com interseção estrutural. Apesar disso, conforme experimentalmente mostrado em [Li01], os algoritmos de interseção estrutural ainda conseguem ser mais eficientes do que os métodos de busca navegacional.

Em contraste, com o objetivo de eliminar o uso de interseções estruturais bem como adotar uma estrutura de dados mais compacta do que a empregada no Dataguide, o ViST tem o documento por inteiro como unidade básica de indexação, evitando decompor consultas com ramificação. Além disso, a árvore de sufixos empregada no ViST permite representar os elementos indexados de uma forma compacta, reduzindo o tempo de acesso de recuperação das informações indexadas. Sendo assim, estas duas propriedades do ViST implicam em um desempenho geral eficiente para o processamento de consultas.

Seguindo esta mesma direção, a técnica BranchGuide também não emprega interseção estrutural para uma classe de expressões com ramificação suportada. Além disso, não emprega uma estrutura de dados navegacional no armazenamento de índice. A técnica BranchGuide adota um índice invertido para armazenamento dos índices, mapeando cada caminho indexado para uma listagem de resultados. Assim, ao buscar por um determinado caminho não é necessário processar interseções estruturais bem como não é necessário navegar sob uma estrutura de caminhos em memória. Adotando um índice invertido é possível oferecer um mecanismo eficiente para recuperar diretamente uma lista de resultados para um determinado caminho [Manning08], e assim, prover um mecanismo escalável para o processamento de consultas com ramificação.

### 3 A Técnica BranchGuide

A linguagem XPath [W3C99] possibilita localizar partes (nós de elementos ou atributos) de documentos XML (no Apêndice A.2 são apresentadas resumidamente algumas funcionalidades da linguagem). Especificamente, XPath é uma linguagem de consulta básica que permite selecionar nós contidos em documentos XML seguindo um determinado padrão [Gou07].

Em alguns contextos, o uso de todas as funcionalidades providas pela linguagem XPath não é necessária. Por exemplo, temos que a *NEXI (Narrowed Extended XPath)* [Trotman05] é uma linguagem de consulta baseada no XPath e estendida com funcionalidades de ranqueamento baseado sob a frequência de valores textuais, mas, com funcionalidades propositalmente limitadas em relação ao XPath. Isto ocorre motivado pelo fato que quando os usuários não têm uma consciência preliminar das características estruturais da coleção indexada então temos que no processo de consulta estes utilizam na maioria das vezes somente o eixo (Apêndice A.2) *ancestral-descendente* (`//`) do XPath. Ao utilizar o eixo ancestral-descendente é possível definir relações de consulta em que um nó XML é descendente, mas não necessariamente filho de outro nó. Assim, informações estruturais podem ser parcialmente omitidas no processo de consulta.

A motivação para o NEXI restringir o XPath é que os usuários têm dificuldades para expressar suas necessidades de informação e tendem a gerar erros semânticos no processo de consulta. Estes enganos no processo de consulta são uma consequência do fato que previamente o usuário tem um conhecimento incompleto da estrutura dos documentos [Kamps06]. Uma alternativa para lidar com este problema é o projeto de interfaces gráficas de usuário que expõem explicitamente a estrutura da coleção de documentos semi-estruturados, permitindo ao usuário especificar os elementos de interesse ao mesmo tempo em que é guiado pela interface gráfica [Manning08]. Esta abordagem é denominada de *interface gráfica de consulta dirigida via esquema*. Como neste caso as restrições estruturais são bem definidas no

processo de construção de consultas, ao invés de utilizar o eixo ancestral-descendente ( // ) do XPath, os usuários devem construir consultas utilizando o eixo *pai-filho* ( / ) do XPath, que permite definir relações de consulta em que um nó é filho direto de outro nó, de forma a expressar precisamente as informações dos nós buscados.

A técnica BranchGuide é projetada para ser empregada em sistemas de busca para dados semi-estruturados que idealmente disponibilizam uma interface gráfica de consulta dirigida via esquema, assim, através de consultas expressas através do eixo pai-filho ( / ) do XPath podem ser recuperados documentos de uma fonte de dados.

Considerando que o XPath é uma linguagem de consulta que permite selecionar nós que seguem um determinado padrão, uma abordagem de indexação é preservar o registro da ocorrência de determinados padrões nos documentos indexados. Seguindo esta direção, algumas abordagens de indexação de dados semi-estruturados suportam uma seleção de determinadas classes de expressões de consulta para serem avaliadas de forma privilegiada. A ideia é representar e indexar todos os caminhos que seguem um determinado padrão. Esta abordagem pode aprimorar o tempo de recuperação bem como diminuir os custos de armazenamento de índice [Weigel02]. Apesar destes benefícios, ao selecionar uma classe de caminhos para indexação implica-se em uma limitação em relação ao conjunto total de expressões que são suportadas pelo sistema de recuperação. Assim, existe um compromisso entre a capacidade de um sistema poder processar consultas mais expressivas e os requisitos de armazenamento do índice e de tempo de processamento das consultas [Kamps06].

Já que a técnica BranchGuide é projetada para ser empregada em sistemas de busca que disponibilizam uma interface gráfica de consulta dirigida via esquema, então o conjunto de expressões suportadas pela técnica compreendem expressões expressas através do eixo pai-filho ( / ). Neste sentido, com o objetivo de reduzir os custos de processamento de consulta e tamanhos de arquivos índice, a técnica BranchGuide adota uma abordagem de seleção de uma classe de caminhos simples e com ramificação os quais são extraídos a partir da ocorrência da relação de eixo pai-filho ( / ) entre os nós da fonte de dados.

Especificamente, BranchGuide é uma técnica de indexação exata (sem erros de precisão) para uma definida classe de expressões de caminhos com ramificação. Ao invés de uma solução generalista que permite processar todos os eixos do XPath, BranchGuide adota um padrão de seleção para um conjunto básico de expressões de caminhos com ramificação. O conjunto de expressões de caminhos selecionados pode ser empregado em sistemas de busca que não demandem o exagerado e abundante poder de expressividade de consulta

definido na especificação original do XPath, mas ao invés disso, em sistemas de busca que estão em conformidade com os seguintes requisitos:

*i)* A unidade de recuperação adotada é o documento XML ao invés de particulares elementos e atributos XML. O objetivo é recuperar uma entidade que é descrita através de um particular documento XML e não através de partes de um documento XML.

*ii)* Um padrão bem definido para seleção de caminhos com ramificação é adotado para evitar o retorno de falsos positivos no processamento de expressões de caminhos com ramificação que pertencem a tal classe.

*iii)* Na busca por documentos, restrições sob valores textuais de atributos podem ser impostas. Além disso, por meio de um mecanismo de recuperação baseado em operadores lógicos, o usuário pode procurar por documentos que contêm somente uma porcentagem de todas as restrições definidas.

*iv)* Do ponto de vista do modo de construção de consultas, temos que as informações estruturais são um critério preciso de necessidade de informação do usuário de forma que as consultas são expressas através de uma interface gráfica dirigida via esquema. Portanto, o processamento de expressões de consulta do eixo pai-filho ( / ) do XPath é suportado.

*v)* É demandado o armazenamento de índices de baixo custo bem como um eficiente mecanismo de processamento de consultas.

A definição dos requisitos *ii*, *iii* e *iv* implicam diretamente na redução de expressividade da linguagem de consulta suportada pela técnica de indexação BranchGuide em relação a todas as funcionalidades da linguagem XPath. Apesar disso, tais requisitos são definidos a fim de causar um impacto no custo de processamento de consultas e de armazenamento de índice, buscando assim atender ao critério *v*. Neste sentido, neste capítulo, uma nova técnica de indexação, denominada BranchGuide, que visa atender os requisitos definidos é proposta.

Inicialmente, na Seção 3.1, o padrão de seleção de caminhos proposto é descrito e detalhado. Já na Seção 3.2, o método de indexação que possibilita que os caminhos que adotam o padrão de seleção proposto sejam extraídos dos documentos da fonte de dados e indexados é definido. Além do método de indexação é necessário definir um método de construção e processamento de consultas para assim habilitar o acesso aos índices e prover a recuperação de documentos, neste sentido, o modelo de recuperação de informação adotado na técnica BranchGuide é proposto e detalhado na Seção 3.3. Por fim, considerações finais sobre a técnica BranchGuide são discutidas na Seção 3.4.

### 3.1 Padrão de Seleção de Caminhos

XML é baseado em um modelo hierárquico: cada elemento XML pode conter um valor textual ou outro elemento filho. Isto é, as associações entre elementos são especificadas através da hierarquia. Além disso, elementos XML podem conter atributos. Um atributo é um par *nome-valor* em que informações descritivas sobre o elemento são armazenadas [Harold01].

As propriedades de um documento XML podem ser restritas através de regras especificadas. Por exemplo, a cardinalidade de elementos filhos é uma importante propriedade porque tem um efeito direto na ocorrência de falsos positivos quando utilizada uma técnica de indexação de caminhos simples. Baseado no exemplo ilustrado na Figura 15, *Editora*, *Livro* e *Autor* são três conceitos expressados através de elementos relacionados de mesmo nome. A Figura 15 ilustra diferentes casos de restrições de regras de cardinalidade.

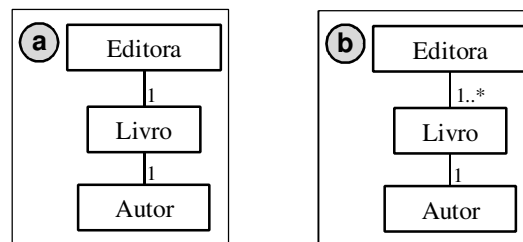


**Figura 15: Regras de cardinalidade**

Na Figura 15a, cada *Editora* deve publicar somente um único *Livro*, o qual por vez deve ter exatamente um único *Autor*. Neste caso, uma técnica de indexação baseada em caminhos simples não gera falsos positivos.

Por outro lado, na Figura 15b, cada *Editora* pode ter vários *Livros* publicados, e cada *Livro* deve ter um único *Autor*. Neste caso, levando em conta que cada editora tem vários livros publicados, adotando uma técnica de indexação baseada em caminhos simples não é possível saber quais são precisamente as *Editoras* que publicam um determinado *Livro* escrito por um determinado *Autor* (ocasionado pelo problema discutido na Seção 2.1.2). A fim de tratar este problema, é necessário indexar as relações de cardinalidade entre os elementos (juntamente com respectivas informações dos atributos que pertencem a cada elemento).

A técnica BranchGuide obtém proveito da indexação de uma bem definida classe de caminhos com ramificação. Este conjunto de caminhos selecionados captura as relações de cardinalidade entre elementos juntamente com as informações dos atributos pertencentes a cada elemento. Mais especificamente, o padrão de seleção de caminhos é formalmente

especificado através do uso da notação XPath [W3C99] e é dada pela expressão regular mostrada na Figura 16, que representa uma navegação de caminho composta por uma ilimitada sequência de elementos XML, em que cada um pode conter no máximo um dado atributo XML com respectivo valor.

$$P = (/elemento([@atributo="valor"])^{0..1})^+$$

**Figura 16: Padrão de seleção de caminhos proposto**

Deve ser notado que o padrão de seleção de caminhos  $P$  permite que todos os caminhos simples e um conjunto substancial de caminhos com ramificação sejam selecionados para indexação. Sendo assim, o padrão permite selecionar caminhos mais expressivos do que caminhos simples.

Por exemplo, o padrão de seleção  $P$  permite selecionar tanto caminhos simples tais como  $p_1$  e  $p_2$  (mostrados na Figura 17), bem como, o caminho com ramificação  $p_3$  (mostrado na Figura 17). A seleção do caminho com ramificação  $p_3$  possibilita que as dependências entre o elemento XML *Livro* com nome “*Hamlet*” e o elemento XML *Autor* com nome “*Shakespeare*” sejam individualmente indexadas.

$$\begin{aligned} p_1 &= /editora/livro[@nome="Hamlet"] \\ p_2 &= /editora/livro/autor[@nome="Shakespeare"] \\ p_3 &= /editora /livro[@nome="Hamlet"]/autor[@nome="Shakespeare"] \end{aligned}$$

**Figura 17: Exemplos de caminhos selecionados**

A próxima seção detalha como os caminhos que satisfazem o padrão de seleção  $P$  são extraídos a partir da fonte de dados e indexados utilizando a técnica BranchGuide.

### 3.2 Indexação

A técnica BranchGuide adota um índice de mapeamento de caminhos com ramificação, que satisfazem o padrão de seleção  $P$ , para um conjunto de identificadores de documentos nos quais ocorrem um dado caminho indexado. A entrada do índice é expressa através de uma consulta por um determinado caminho procurado.

A fim de apresentar formalmente a técnica de indexação proposta, uma notação baseada em grafos é adotada. Uma fonte de dados indexada é representada pelo grafo de índice  $G' = (V, E, \text{rótulo}, \text{valor}, \text{tipo}, \text{RAIZ})$ , no qual  $\text{RAIZ}$  é a raiz de  $G'$ . O conjunto de vértices de  $G'$  é representado por  $V$  e o conjunto de arestas é representado por  $E$ . Cada vértice

pode representar um elemento ou atributo XML. Com isto, a função *tipo* mapeia cada vértice de  $V$  para exclusivamente um dos seguintes tipos: atributo ou elemento.

Por sua vez, a função *rótulo* mapeia cada aresta de  $E$  para o nome de um respectivo elemento ou atributo XML.

Já a função *valor* é uma função que mapeia cada vértice  $v \in V$  para uma lista de identificadores de documentos. Formalmente, cada caminho é uma sequência de vértices ligados a partir do vértice *RAIZ* terminando em um dado vértice  $v \in V$ . Como o último vértice de um caminho nunca aparece como último vértice de qualquer outro caminho, então, cada caminho é diretamente associado ao seu último vértice. Assim, para uma determinada lista  $l$  de identificadores de documentos mapeada pela função *valor*:  $v \rightarrow l$  temos que o caminho formado a partir do vértice *RAIZ* até o vértice  $v$  ocorre em todos os documentos listados pela a lista  $l$  de identificadores de documentos.

Vale ressaltar que, para cada identificador de documento, a técnica de indexação possibilita associar informações auxiliares dependentes do modelo de recuperação adotado. Estas informações auxiliares podem ser utilizadas na recuperação de documentos. Em outras palavras, a técnica possibilita que informações adicionais sejam armazenadas, e a função *valor* é definida para permitir este ponto de extensão.

Adicionalmente, através da função *valor*, se o vértice é um atributo XML ou um elemento XML com valor textual, dentre as informações mapeadas por *valor* também estão contidos o valor textual do elemento ou textual do atributo.

Perceba que, enquanto a função *rótulo* mapeia as informações das arestas ( $E$ ), a função *valor* mapeia as informações dos vértices ( $V$ ).

A fim de indexar caminhos com ramificação que conformam com o padrão de seleção de caminhos  $P$ , a técnica de indexação proposta gera o grafo de índice  $G'$  a partir da fonte de documentos. Em  $G'$ , todo caminho indexado é representado exatamente uma única vez e não existe nenhum caminho replicado no grafo.

Para exemplificar a extração de caminhos, considere a seguinte fonte de dados composta por documentos descrevendo o catálogo de livro de editoras, como ilustrado na Figura 18. A técnica BranchGuide gera o grafo de índice  $G'$  ao aplicar em sequência as seguintes regras sob os documentos da fonte de dados:

**Documento 1:**

```

<editora nome="SparkNotes">
  <livro nome="Hamlet">
    <autor nome="Shakespeare"/>
  </livro>
</editora>

```

**Documento 2:**

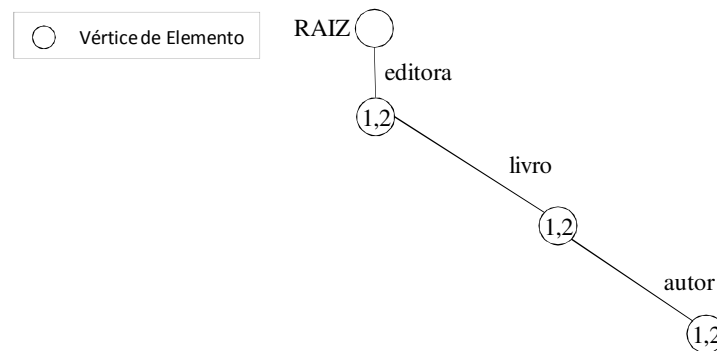
```

<editora nome="Simon&Schuster">
  <livro nome="Hamlet">
    <autor nome="Shakespeare"/>
  </livro>
  <livro nome="Othello">
    <autor nome="Shakespeare"/>
  </livro>
</editora>

```

**Figura 18: Um exemplo de fonte de dados**

**Regra I:** Cada caminho extraído a partir de um documento consiste de uma sequência de vértices separados por arestas que correspondem à respectiva estrutura hierárquica dos elementos XML do documento, iniciando a partir do elemento *RAIZ* até o último elemento do caminho. Para cada caminho, o identificador de documento (que contém uma ocorrência do caminho) é incluído na lista de identificadores de documentos associados com cada vértice. Já cada aresta incidente ao vértice correspondente a um elemento XML é rotulada com o nome do respectivo elemento XML. Por exemplo, na Figura 19, o caminho */editora/livro/autor* que ocorre nos *Documentos 1 e 2* (mostrados na Figura 18) é derivado a partir desta regra.

**Figura 19:  $G'$  após aplicação da regra I**

**Regra II:** Para cada atributo de um determinado elemento XML existente em um documento, uma nova aresta e vértice folha são criados e conectados ao vértice que corresponde ao respectivo elemento XML. Além disso, o identificador do documento em que ocorre o atributo deve ser incluído na lista de identificadores de documentos do vértice folha e a aresta que incide no vértice folha é rotulada com o nome do atributo XML. Como exemplo, o caminho */editora/livro/autor[@nome="Shakespeare"]* que ocorre nos *Documentos 1 e 2* (como mostrado na Figura 18) é adicionado em  $G'$  de acordo com esta regra e o resultado obtido é mostrado na Figura 20.

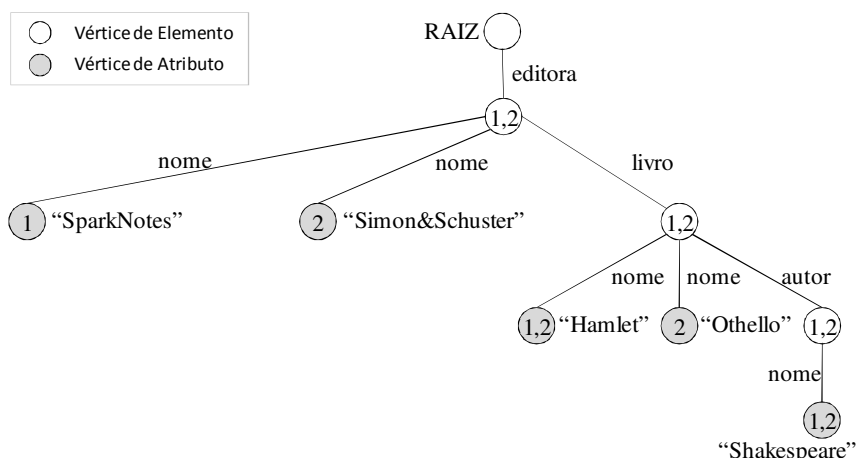


**Figura 20:  $G'$  após aplicação da regra II**

**Regra III:** Para cada vértice de elemento em  $G'$  relativo a um elemento que contenha elementos filhos e atributos simultaneamente, temos que cada subárvore enraizada por cada elemento filho deve ser recriada abaixo de cada vértice de atributo. Ao recriar a subárvore, somente os caminhos com ramificação que ocorrem nos documentos da fonte de dados devem ser incluídos na subárvore recriada. A Figura 21a mostra uma partição de um grafo de índice em que um dado *elemento pai* tem  $n$  atributos e um *elemento filho*. Neste caso, a subárvore enraizada pelo *elemento filho* deve ser recriada abaixo de cada um dos  $n$  atributos gerando  $n$  ramificações indexadas. Na Figura 21b, a notação de arestas duplas ressalta a extração de caminhos de ramificação indexados que satisfazem o padrão de seleção de caminhos  $P$ .

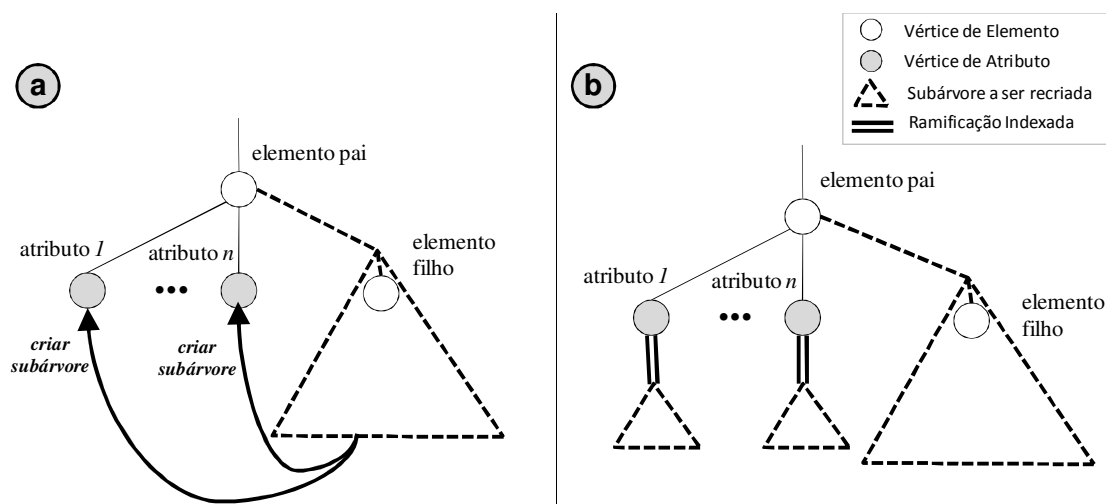


**Figura 21: Regra III**

Por exemplo, observe que na fonte de dados ilustrada na Figura 18 existem elementos que estão na condição de possuir simultaneamente atributos e elementos filhos. Primeiramente, o elemento *livro* contém o elemento filho *autor* e instâncias do atributo *nome*.





[illegible]

**Figura 24: Caminhos extraídos a partir da fonte de dados**

com ramificação contém o conjunto exato de documentos em que o caminho ocorre.

usuário de forma estas sejam mapeadas para caminhos que satisfazem o padrão de seleção de caminhos indexados, com isto é possível habilitar o acesso aos índices e prover a descoberta

de documentos. Neste sentido, o modelo de recuperação de informação adotado na técnica BranchGuide é discutido na seção seguinte.

### 3.3 Recuperação de Informação

Inicialmente é importante definir as diferenças clássicas entre *recuperação de dados* e *recuperação de informação*. De acordo com [Rijsbergen79], uma diferença essencial diz respeito ao que é procurado e ao que é respectivamente encontrado pelo usuário do sistema de recuperação. Em recuperação de dados, o usuário procura por objetos que contém atributos necessários e suficientes para pertencer a uma determinada classe. Portanto, o usuário deve buscar objetos especificando exatamente um conjunto de atributos necessários.

Em contrapartida, em recuperação de informação o usuário procura por uma classe de objetos que contém somente uma porcentagem de todos os atributos especificados. Assim, o usuário poderia encontrar objetos *similares* aos que estava inicialmente procurando. Adotar uma abordagem de recuperação de informação é importante, pois em geral, antes de iniciar as atividades de consulta, o usuário tem somente uma noção parcial do que está procurando. Sendo assim, quando o usuário inspeciona os resultados providos pelo sistema de recuperação, estes resultados podem satisfazer as necessidades de informação do usuário, bem como, o usuário pode redefinir sua necessidade de informação inicial.

Nesta dissertação, quando mencionado *recuperação* estamos tratando de recuperação de informação e não de recuperação de dados. Neste sentido, para um melhor entendimento do leitor, o conceito *recuperação de dados semi-estruturados* deve ser entendido como: recuperação de informação a partir de fonte de dados semi-estruturados.

Em um sistema de recuperação de dados semi-estruturados em que objetos são descritos utilizando o modelo XML temos que as relações estruturais entre elementos dos documentos provêm o valor semântico para os valores textualmente descritos. Do ponto de vista de recuperação, as expressões de caminho permitem que tanto informações semânticas quanto textuais possam ser expressas através de consultas. Desta forma, em um *sistema de busca* (Seção 1.3) que adota a técnica BranchGuide, uma linguagem de expressões de caminho (tal como o XPath) é adotada para possibilitar ao usuário a construção de consultas. Assim, critérios estruturais e textuais sob o conteúdo XML podem ser definidos na recuperação de objetos. Assim, o *motor de busca* (Seção 1.3) do sistema de busca provê métodos de tratamento de similaridade de forma a conduzir ações de processamento necessárias para acessar o arquivo de índice e retornar resultados para o usuário.

Neste contexto, a técnica BranchGuide possibilita que consultas com um nível controlado de similaridade sejam processadas. Para tal, a técnica BranchGuide provê meios para decompor consultas mais complexas com operadores de controle de similaridade em expressões com ramificação que podem ser buscadas no índice.

Mais especificamente, o controle de similaridade é definido em função de operadores lógicos de *disjunção* (*OR*) e *conjunção* (*AND*). Por exemplo, suponha que o usuário está procurando *por editoras que publiquem o livro “Hamlet” reeditado no ano de “2010” e/ou escrito pelo autor “Shakespeare”*. Uma expressão de consulta que representa esta necessidade de informação é mostrada na Figura 25.

/editora/livro[ @nome=“Hamlet” ⊗ @ano=“2010”]/autor[ @nome=“Shakespeare”]

Onde ⊗ é um operador de conjunção (*AND*) ou disjunção (*OR*).

#### Figura 25: Controle de similaridade através de operadores

É importante ressaltar que o usuário pode especificar o grau de similaridade dos resultados definindo explicitamente o operador ⊗ como sendo *AND* ou *OR*, bem como, não indicando o operador de forma que o motor de recuperação retorne itens com diversos graus de similaridade. No caso do usuário **não** definir explicitamente os operadores de similaridade, primeiramente, é gerada uma expressão de consulta mais restrita que utiliza o operador *AND*. Posteriormente, é gerada uma segunda expressão de consulta, sendo esta menos restrita, o qual utiliza o operador *OR*. Para concluir o processamento, o resultado da execução de cada uma das duas consultas é concatenado em uma única lista, que então é retornada ao usuário. Na lista resultante, os itens mais restritos (mais similares) aparecem no topo da lista, enquanto que os itens menos restritos (menos similares) aparecem na base da lista.

A técnica BranchGuide não indexa diretamente por um método força-bruta todas as permutações de conjunção/disjunção existentes nos documentos. Desta forma, consultas com operadores lógicos devem ser decompostas em expressões de ramificação indexadas, e posteriormente, através de um processamento adicional os resultados são combinados de forma que são retornados os objetos que satisfazem a consulta original.

Por exemplo, considere a expressão da Figura 25. Definindo o operador ⊗ como o operador lógico *AND*, a expressão de consulta final é mostrada Figura 26. Para processar tal expressão, ela é decomposta em duas expressões com ramificação que também são mostradas na Figura 26. As duas expressões são individualmente buscadas no índice, e, para cada



Como apontado por Goldman [Goldman97], no pior caso, uma técnica que indexa caminhos simples provê uma complexidade de armazenamento linear. Entretanto, esta propriedade não é verdadeira para técnicas que indexam caminhos com ramificação. Por exemplo, como pode ser visto na Figura 27, a aplicação da técnica BranchGuide conduz para uma complexidade de armazenamento não linear, isto é, uma complexidade de armazenamento maior do que a imposta pelo Dataguide. Esta propriedade de complexidade de armazenamento não linear da técnica BranchGuide implica em um crescimento não desejado do tamanho dos arquivos de índice gerados após o processo de indexação.

Consequentemente, o sucesso da técnica BranchGuide depende em adotar ajustes no projeto de armazenamento do índice de forma que isto reduza a complexidade de armazenamento dos índices gerados, mas ao mesmo tempo em que não se introduza redução de precisão nem aumento nos custos de processamento de consultas.

Neste sentido, no projeto da técnica BranchGuide, ajustes são propostos a partir de propriedades observadas na geração do grafo de índice  $G'$ . No capítulo seguinte, as propriedades observadas a partir do grafo de índice  $G'$  bem como os ajustes definidos no projeto de armazenamento de índice são discutidos.

## 4 Projeto da Técnica BranchGuide

Conforme apresentado anteriormente na Seção 3.3, a técnica BranchGuide possibilita que consultas com operadores lógicos sejam decompostas em expressões com ramificação que podem ser buscadas no arquivo de índice. Uma simples abordagem para prover o processamento de operadores lógicos seria a indexação de todo o conjunto possível de permutações existentes que empregam relações lógicas entre todos os valores de atributos da fonte de dados. Entretanto, esta não é uma boa solução, pois gera uma quantidade volumosa de informação a ser indexada.

Assim, um método mais adequado para prover o processamento de operadores lógicos é decompor as expressões de consulta com operadores lógicos para um subconjunto de expressões de consulta que conformam com a interface de processamento do índice. No caso da técnica BranchGuide, as expressões devem satisfazer o padrão de seleção de caminhos adotado. Dado este problema, a Seção 4.1 detalha o projeto do mecanismo de processamento de consultas adotado na técnica BranchGuide. Este mecanismo permite processar expressões com operadores lógicos.

Além de evitar a indexação de relações lógicas, a técnica BranchGuide indexa uma classe de caminhos com o objetivo de amortizar a quantidade de informação indexada. Isto é, ao invés de indexar todos os possíveis caminhos que podem ser extraídos da fonte de dados, a técnica BranchGuide indexa uma classe bem definida de caminhos com ramificação.

É importante notar que um índice Dataguide (que indexa somente os caminhos simples) está contido no índice BranchGuide. Sabendo que o Dataguide tem no pior caso uma complexidade de armazenamento linear [Goldman97], então o BranchGuide tem uma complexidade de pior caso maior do que linear já que indexa caminhos com ramificação além dos caminhos simples. Esta complexidade de armazenamento da técnica BranchGuide pode conduzir para tamanhos de arquivos indesejados e geralmente maiores do que o tamanho da fonte de dados. A fim de amenizar este efeitos indesejados, a técnica BranchGuide adota um

projeto de índice que reduz a quantidade de informação armazenada, mas sem introduzir perda de precisão ou aumento no tempo de processamento de consulta. Desta forma, o projeto de índice proposto permite que a técnica BranchGuide possa ser adotada na prática em sistemas de recuperação de informação. O projeto de índice proposto é discutido na Seção 4.2.

#### 4.1 Projeto do Mecanismo de Processamento de Consultas

Gerar permutações de todas as conjunções e disjunções a partir da fonte de dados e indexá-las não é uma abordagem ideal, pois este método exaustivo conduz para um conjunto grande de soluções indexadas, gerando assim, uma quantidade demasiada de informações armazenadas.

Neste sentido, a fim de tratar este problema, na técnica BranchGuide é definido um mecanismo para o tratamento de operadores lógicos. Ao invés de indexar informações de forma exaustiva, o mecanismo faz uso de processamento adicional em tempo de consulta e do armazenamento de informações auxiliares no índice. É importante ressaltar que, para definir o método de processamento de consultas recorre-se à definição original da técnica de indexação (Seção 3.2) que possibilita associar informações auxiliares para cada identificador de documento indexado, e tais informações podem ser utilizadas na recuperação de documentos.

Uma expressão de consulta com ramificação e operador lógico AND é mostrada na Figura 28a e representada através de um grafo na Figura 28b.

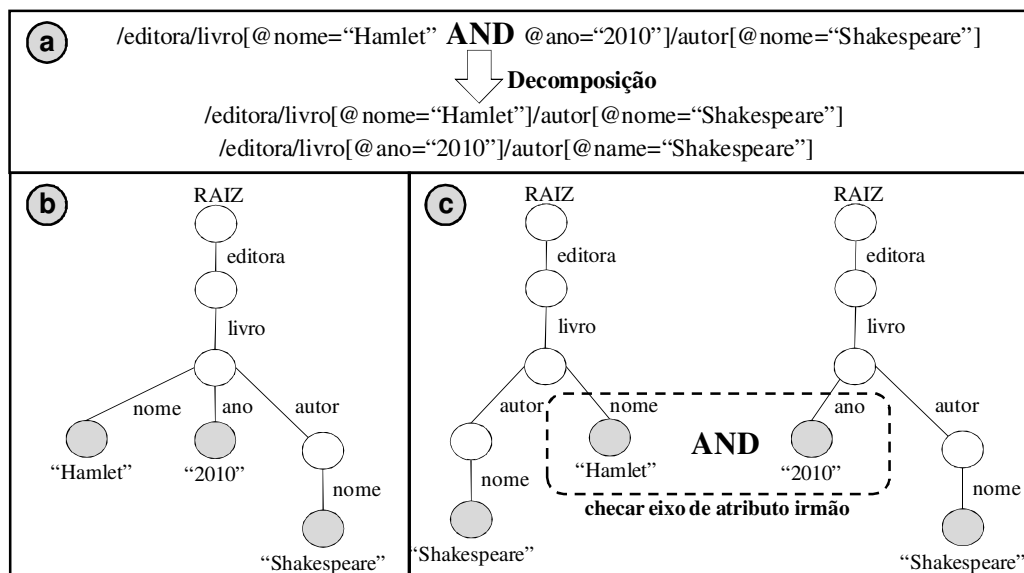


**Figura 28: Checagem de eixo de atributo irmão**

Como a expressão de consulta mostrada na Figura 28a não satisfaz o padrão de seleção suportado pela técnica BranchGuide, a expressão tem que ser decomposta nas duas subconsultas listadas na Figura 28a (e representadas através de grafos ilustrados na Figura

28c) que conformam com o padrão de seleção de expressões suportado pela técnica BranchGuide. Assim, a expressão da Figura 28b é decomposta em duas subexpressões de ramificação suportadas pela técnica proposta, com isto o índice é acessado duas vezes e os dois resultados encontrados devem ser combinados a fim de gerar o resultado final. A combinação deve ter como resultado somente os documentos tal que existam a ocorrência das duas expressões de ramificação da Figura 28c, e, além disso, os atributos *nome* e *ano* pertençam a um mesmo elemento *livro*.

Neste contexto, define-se que *dois atributos são irmãos quando pertencem ao mesmo elemento de um documento*, desta forma, para checar a condição de conjunção de expressão mostrada na Figura 28c é necessário checar o *eixo de atributo irmão* entre atributos. Por exemplo, a condição de verificação de que os atributos *nome* e *ano* são irmãos é visualmente representada na Figura 28b através de uma borda pontilhada.

Definidos tais conceitos, na próxima subseção é detalhado o método de identificação de eixo de atributo irmão adotado na técnica de indexação BranchGuide, este método possibilita checar se dado dois atributos, estes são irmãos.

#### 4.1.1 Método de Identificação de Atributos Irmãos

Para checar se quaisquer atributos são irmãos temos que para todo caminho com vértice final do *tipo atributo* (definido na Seção 3.2) além da lista de identificadores de documentos também são armazenadas informações auxiliares. Estas informações auxiliares são utilizadas durante tempo de processamento de consulta para checar se a conjunção de atributos é ou não verdadeira e assim, possibilitar o controle de similaridade na recuperação de informações através do emprego de operadores lógicos.

Neste sentido, define-se que as informações auxiliares acrescentadas no índice consistem em um conjunto de *códigos de eixo de atributo irmão*. A atribuição dos códigos é individualizada por documento, e inicia a partir do valor de código 1 até o valor máximo que é a quantidade de ocorrências de diferentes eixos de atributo irmão no documento. Além disso, adota a seguinte regra: *Dois atributos têm o mesmo valor de código de eixo de atributo irmão, se, e somente se, eles forem atributos irmãos*.

A Figura 29 mostra um documento em que os códigos atribuídos estão destacados, é importante ressaltar que neste caso existem cinco diferentes eixos de atributo irmão no documento de forma que a atribuição varia de forma crescente de 1 até 5.

| eixo de atributo irmão |                                   |
|------------------------|-----------------------------------|
| 1                      | <editora nome="Simon&Schuster">   |
| 2                      | <livro nome="Hamlet" ano="2010">  |
| 3                      | <autor nome="Shakespeare"/>       |
|                        | </livro>                          |
| 4                      | <livro nome="Othello" ano="2010"> |
| 5                      | <autor nome="Shakespeare"/>       |
|                        | </livro>                          |
|                        | </editora>                        |

**Figura 29: Exemplo de atribuição de códigos de eixo de atributo irmão**

Empregando códigos de eixo de atributo irmão podemos checar se condições de conjunção são verdadeiras ou falsas. Por exemplo, no processamento da expressão de consulta com operador de conjunção `/editora/livro[@nome="Hamlet" AND @ano="2010"]` sob o documento ilustrado na Figura 29, temos que esta expressão é decomposta nas duas subexpressões `/editora/livro[@nome="Hamlet"]` e `/editora/livro[@ano="2010"]`. A primeira subexpressão retorna o código de eixo de atributo irmão 2 (conforme indicado na Figura 29), já a segunda subexpressão também retorna o código de eixo de atributo irmão 2 (conforme indicado na Figura 29). Já que os dois respectivos códigos de eixo de atributo irmão são iguais, então o documento ilustrado na Figura 29 satisfaz a consulta com conjunção do exemplo. Em outras palavras, os atributos *nome* e *ano* são irmãos, de forma que a conjunção da consulta é verificada positivamente e o documento satisfaz a consulta.

A operação de combinação empregada no processamento de operadores de conjunção consiste em realizar a interseção de conjuntos de pares ordenados (*identificador do documento, código de eixo de atributo irmão*). Por exemplo, supondo que o documento da Figura 29 é identificado por  $D$ , então, a consulta pelo caminho `/editora/livro[@nome="Hamlet"]` retorna o conjunto  $R_1 = \{(D, 2)\}$ . Já a consulta pelo caminho `/editora/livro[@ano="2010"]` retorna o conjunto  $R_2 = \{(D, 2), (D, 4)\}$  pois o atributo `@ano="2010"` ocorre duas vezes no documento  $D$  (Figura 29). Com isto, o resultado do processamento da consulta `/editora/livro[@nome="Hamlet" AND @ano="2010"]` é dado por  $R = R_1 \cap R_2 = \{(D, 2)\}$ , ou seja como  $(D, 2) \in R$  então o documento  $D$  satisfaz a consulta.

A checagem de códigos de eixo irmão é necessária somente no processamento da conjunção. No caso da disjunção, por exemplo, a expressão de consulta `/editora/livro[@nome="Hamlet" OR @ano="2010"]` também é decomposta nas duas

subexpressões */editora/livro[@nome="Hamlet"]* e */editora/livro[@ano="2010"]*, mas, neste caso, qualquer documento que satisfaça qualquer uma das duas subexpressões é um resultado positivo. Logo não é necessário checar se os atributos *nome* e *ano* são irmãos.

Vale ressaltar que, como de acordo com os requisitos definidos no Capítulo 3, a técnica BranchGuide deve prover um mecanismo eficiente de processamento de consultas e gerar tamanhos de índice de baixo custo. A fim de alcançar estes objetivos, o método de identificação de atributos irmãos adotado na técnica BranchGuide tem as seguintes características: i) *O armazenamento de informações auxiliares para checagem do eixo de atributo irmão emprega técnicas de compressão sem perda de precisão.* ii) *Na recuperação, o processamento de combinação de informações auxiliares tem tempo de execução linear.*

As técnicas de compressão empregadas no armazenamento dos códigos de eixo de atributo irmão (listadas e detalhadas na Seção 5.2.4) são eficientes para não degradar o desempenho do sistema em tempo de recuperação ao mesmo tempo em que oferecem o benefício de reduzir o tamanho dos arquivos de índice através da compressão sem perdas.

Já em relação ao custo da operação de combinação de resultados empregada na execução de expressões com operadores lógicos é importante destacar que esta não tem o mesmo custo do uso de uma operação de interseção estrutural.

Mais especificamente, a operação de interseção empregada na técnica de indexação BranchGuide difere de uma operação de interseção estrutural pelo seguinte fato: em geral, identificadores de elementos utilizados em técnicas de interseção estrutural não têm uma função de ordem definida, desta forma na execução de uma operação de interseção estrutural entre dois conjuntos  $S_1$  e  $S_2$  de identificadores de elementos de um dado documento, para checar relações de eixo temos que todo item de  $S_1$  é comparado com todos os itens de  $S_2$ , neste sentido, implicando em uma operação de complexidade de execução dado por  $\mathcal{O}(n^2)$ .

Em contrapartida, dois conjuntos  $S'_1$  e  $S'_2$  de códigos de eixo de atributos irmão têm uma função de ordem definida (ordem decimal). Devido a este fator, pode-se empregar um simples e efetivo algoritmo de combinação de resultados que mantém ponteiros para duas listas (com os conjuntos  $S'_1$  e  $S'_2$  ordenados), o qual percorre as duas listas simultaneamente [Manning08] com um tempo de execução linear em função da quantidade de itens dos conjuntos  $S'_1$  e  $S'_2$ . Ou seja, na combinação empregada na técnica BranchGuide temos, no pior caso, uma complexidade de execução linear  $\mathcal{O}(n)$ .

## 4.2 Projeto do Algoritmo de Indexação

É importante mencionar que a adoção de uma técnica de indexação baseada em caminhos gera um problema relacionado à quantidade de informação indexada. Já que o número de caminhos extraídos a partir da fonte de dados pode ser elevado, a tendência é que o tamanho do índice se torne maior do que o tamanho da fonte de dados [Jiang05]. Com o objetivo de reduzir o tamanho de índice, algumas propostas [Goldman97] [Milo99] indexam somente caminhos selecionados dentre o conjunto de todos os caminhos existentes. A desvantagem de tais propostas é a redução de precisão dos índices. Em contrapartida, a indexação de caminhos com ramificação gera um aumento de precisão de índice, mas em compensação, conduz para que o tamanho de índice se torne maior do que o tamanho da fonte de dados.

Como apontado por Goldman [Goldman97], no pior caso, uma técnica de indexação para somente caminhos simples tem um armazenamento de pior caso de complexidade linear. Entretanto, tal propriedade não é verdadeira para técnicas que indexam caminhos com ramificação. Por exemplo, a técnica BranchGuide, proposta nesta dissertação, gera uma complexidade de armazenamento não linear, maior do que a gerada por um Dataguide. Esta propriedade pode causar um crescimento indesejado do tamanho dos arquivos de índice.

Consequentemente, o sucesso da técnica BranchGuide depende de ajustes de projeto para reduzir a complexidade de armazenamento, mas sem introduzir um fator de perda de precisão. Sendo assim, na técnica BranchGuide são empregados ajustes de projeto de índice modelados a partir de propriedades observadas no grafo de índice. Logo, o propósito desta seção é definir o projeto adotado no armazenamento e acesso ao índice, e ressaltar como tais ajustes de projeto de algoritmo impactam nos requisitos de armazenamento.

### 4.2.1 Duplicação de Nós

A aplicação da *Regra III* (Seção 3.2) na extração e indexação de caminhos gera uma grande quantidade de informação sobre os caminhos com ramificação a serem indexados, mais especificamente, a *Regra III* causa um crescimento indesejado da quantidade de vértices que devem ser indexados.

Para facilitar a compreensão do leitor no entendimento deste problema, adota-se um exemplo em que um documento simples é indexado; isto possibilita que o grafo de índice não contenha excesso de informações, e assim, pode-se observar com menor dificuldade os pontos de crescimento do grafo de índice. O *Documento 3* mostrado na Figura 30 descreve uma

*editora* de nome “SparkNotes”, que tem seu escritório localizado na *cidade* de “Oxford”; além disso, esta editora publica dois *livros*: “Hamlet” e “Othello”.

```
<editora nome=“SparkNotes” cidade=“Oxford”>
  <livro nome=“Hamlet”/>
  <livro nome=“Othello”/>
</editora>
```

**Documento 3**

**Figura 30: Documento de exemplo 3**

Para este exemplo, o grafo de índice gerado um passo antes de aplicar a *Regra III* pode ser visto na Figura 31. A *Regra III* é aplicada sob cada vértice de elemento em que as seguintes condições são simultaneamente satisfeitas para o respectivo elemento: i) *O elemento tem ao menos um atributo*; ii) *O elemento tem ao menos um elemento filho*. Assim, cada subárvore abaixo de cada elemento filho é recriada abaixo de cada atributo, mas considerando somente os caminhos com ramificação que ocorrem no documento. Por exemplo, na Figura 31, o elemento *editora* tem os atributos *nome* e *cidade*. Além disso, o elemento *editora* tem um elemento filho denominado *livro*. Portanto, a subárvore abaixo do elemento *livro* (ressaltada na Figura 31 com uma borda pontilhada) deve ser recriada abaixo dos atributos *nome* e *cidade*. Para cada subárvore recriada, esta deve conter somente os caminhos com ramificação que verdadeiramente ocorrem no *Documento 3* (Figura 30).

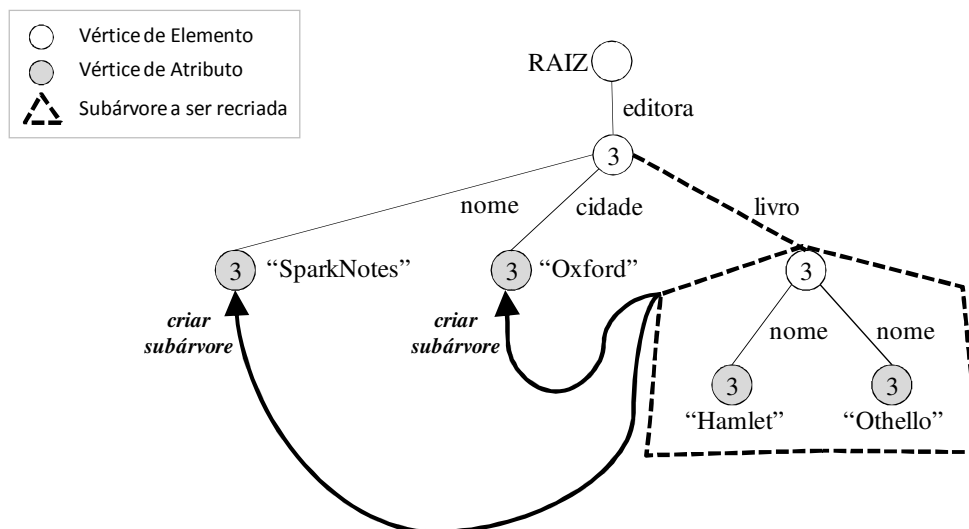


**Figura 31: Criando subárvores através da Regra III**

Após aplicação da *Regra III* na indexação do Documento 3, o grafo de índice obtido é mostrado na Figura 32. Na figura, os caminhos simples estão destacados através de uma borda pontilhada cinza, e este subgrafo corresponde ao Dataguide. Perceba que o grafo de índice

final cresceu e se tornou maior do que o subgrafo do Dataguide. No melhor caso, este crescimento é linear ao tamanho do grafo de dados (isto é, a fonte de entrada), ocorrendo somente quando os documentos da fonte de dados não tem nenhum atributo já que na aplicação da *Regra III* novos vértices são sempre recriados abaixo de atributos.

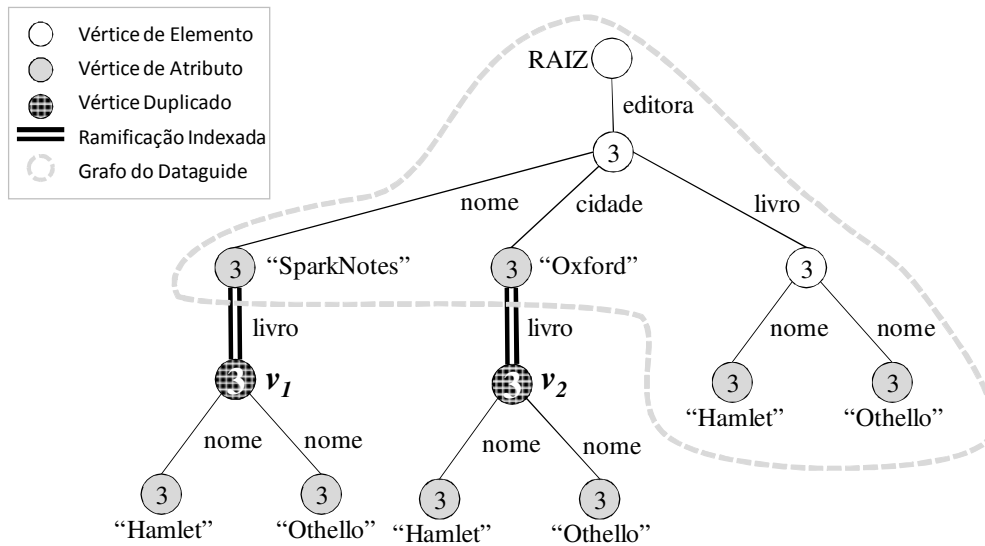


**Figura 32: Grafo de índice  $G'$  obtido após indexação sob o Documento 3**

Como decorrência do aumento de vértices no grafo de índice, temos que a recriação de subárvores pode gerar no grafo de índice a duplicação dos valores mapeados pelos vértices e arestas recriados. Especificamente, uma das propriedades observadas na geração do grafo de índice é que ocorrem condições em que diferentes nós (isto é, nós que correspondem aos vértices finais de caminhos diferentes) mapeiam informações redundantes.

Neste contexto, define-se que dois vértices  $v_1$  e  $v_2$  são *duplicados* quando as três seguintes condições são verdadeiras: i) O valor mapeado pela função rótulo para a aresta incidente em  $v_1$  é igual ao valor mapeado pela função rótulo para a aresta incidente em  $v_2$ . ii) Os valores mapeados pela função valor para o vértice  $v_1$  são iguais aos valores mapeados pela função valor para o vértice  $v_2$ . iii) Os vértices  $v_1$  e  $v_2$  têm a mesma altura (a altura de um vértice é a distância entre o nó RAIZ e o vértice em questão).

De forma mais simplificada, dois vértices são duplicados se tiverem a mesma altura e nome de aresta incidente e, além disso, todos os valores mapeados pelos nós são iguais (estes valores incluem listas de identificadores de documentos, listas de códigos de eixo atributo irmão e valores textuais). Vale ressaltar que a propriedade de duplicação é transitiva, ou seja, se  $v_1$  e  $v_2$  são duplicados e  $v_2$  e  $v_3$  são duplicados, então  $v_1$  e  $v_3$  são duplicados.

Por exemplo, no grafo mostrado na Figura 32, os vértices  $v_1$  e  $v_2$  (destacados com preenchimento em pontilhado) alcançados pelos seguintes caminhos são duplicados: */editora[@cidade="Oxford"]/livro* e *editora[@nome="SparkNotes"]/livro*.

Desta forma, ajustes no projeto do arquivo de índice devem ser adotados para reduzir a duplicação de vértices, mas ao mesmo tempo sem prejudicar o tempo de acesso aos arquivos do índice. O algoritmo que detecta estas duplicações de vértice no processo de indexação e elimina a redundância de dados é detalhado nas próximas seções. Inicialmente, é detalhado o método para amortizar a quantidade de informações de nomes de elementos e atributos, bem como, valores textuais indexados. Por fim, é detalhado o método que reduz a replicação de informações dos resultados de pesquisa, compostos por listas de identificadores e códigos de eixo atributo irmão que são empregados na recuperação de documentos.

#### 4.2.2 Mapeamento de Códigos de Rótulo-Valor

Define-se a *assinatura* para um dado vértice  $v$  do grafo de índice  $G'$  como um par ordenado, em que o primeiro termo corresponde ao *rótulo* da aresta que incide em  $v$ , e por sua vez, o segundo termo corresponde ao valor textual mapeado em  $v$  através da função *valor* ou *vazio* ( $\emptyset$ ) caso nenhum valor textual é mapeado. Por exemplo, na Figura 33, na altura 3 do grafo de índice ilustrado (destacada através de uma linha de corte pontilhada escura), existem quatro vértices  $v_1$ ,  $v_2$ ,  $v_3$  e  $v_4$  indicados. Inspeccionando estes vértices da esquerda para direita temos como resultado as seguintes assinaturas: (*"livro"*,  $\emptyset$ ), (*"livro"*,  $\emptyset$ ), (*"nome"*, *"Hamlet"*) e (*"nome"*, *"Othello"*) para os vértices  $v_1$ ,  $v_2$ ,  $v_3$  e  $v_4$  respectivamente.

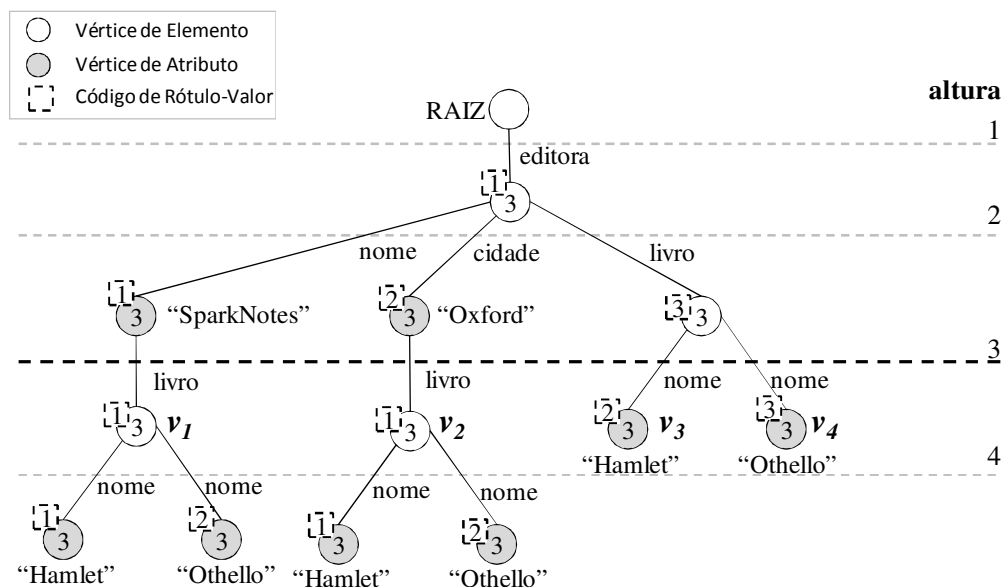


Figura 33: Atribuição de códigos rótulo-valor



valor. Ou seja, o caminho `/editora[@nome="SparkNotes"]/livro[@nome="Othello"]` (na Figura 34) é redefinido através da sequência numérica 1, 1, 1, 2, valores que correspondem sequencialmente ao código de rótulo-valor de cada um dos vértices  $v_5$ ,  $v_6$ ,  $v_7$ ,  $v_8$ .

A partir desta codificação, qualquer caminho extraído a partir de um grafo de índice tem uma respectiva e exclusiva sequência numérica associada. Na Tabela 2 estão listados todos os caminhos e respectivas sequências numéricas extraídas a partir do grafo de índice ilustrado na Figura 34.

**Tabela 2: Caminhos e sequências numéricas**

| Caminho                                                          | Sequência Numérica |
|------------------------------------------------------------------|--------------------|
| <code>/editora</code>                                            | 1                  |
| <code>/editora[@nome="SparkNotes"]</code>                        | 1, 1               |
| <code>/editora[@nome="SparkNotes"]/livro</code>                  | 1, 1, 1            |
| <code>/editora[@nome="SparkNotes"]/livro[@nome="Hamlet"]</code>  | 1, 1, 1, 1         |
| <code>/editora[@nome="SparkNotes"]/livro[@nome="Othello"]</code> | 1, 1, 1, 2         |
| <code>/editora[@cidade="Oxford"]</code>                          | 1, 2               |
| <code>/editora[@cidade="Oxford"]/livro</code>                    | 1, 2, 1            |
| <code>/editora[@cidade="Oxford"]/livro[@nome="Hamlet"]</code>    | 1, 2, 1, 1         |
| <code>/editora[@cidade="Oxford"]/livro[@nome="Othello"]</code>   | 1, 2, 1, 2         |
| <code>/editora/livro</code>                                      | 1, 3               |
| <code>/editora/livro[@nome="Hamlet"]</code>                      | 1, 3, 2            |
| <code>/editora/livro[@nome="Othello"]</code>                     | 1, 3, 3            |

#### 4.2.4 Tratamento da Replicação de Valores de Rótulo e Valores Textuais

Como citado anteriormente (e ilustrado na Figura 32), os vértices  $v_1$  e  $v_2$  são duplicados. Neste contexto, a atribuição de código rótulo-valor permite que ocorrências de condições de replicação sejam tratadas. Especificamente, os valores de rótulo e valor textual de vértices duplicados são armazenados sem replicação, pois ao invés de armazenar os valores de assinatura (*rótulo*, *valor textual*) várias vezes, basta armazenar um mapeamento das assinaturas para códigos de rótulo-valor. Desta forma, no processo de indexação, ao invés de serem armazenados e manipulados os caminhos em forma de *strings*, são armazenados e manipulados as sequências numéricas correspondentes.

Para isto, uma tabela de mapeamento de códigos para cada nível de  $G'$  é gerada e mantida após processo de indexação. Por exemplo, a tabela de mapeamento de códigos gerada para os vértices de altura 3 pertencentes ao grafo de índice ilustrado na Figura 33 é mostrado na Tabela 3. Perceba que existem quatro vértices na altura 3, mas na tabela de mapeamento só existem três entradas pois **não** é necessário armazenar os valores de rótulo (*livro*,  $\emptyset$ ) dos

vértices  $v_1$  e  $v_2$  (Figura 33) duas vezes, ao invés disso, mantém-se uma tabela que mapeia a respectiva assinatura para o código  $l$ .

**Tabela 3: Tabela de mapeamento de códigos para a altura 3**

| Assinatura (rótulo, valor) | Código Rótulo-Valor |
|----------------------------|---------------------|
| ("livro", $\emptyset$ )    | 1                   |
| ("nome", "Hamlet")         | 2                   |
| ("nome", "Othello")        | 3                   |

Vale ressaltar que o método adotado amortiza a quantidade de informações de nomes de elementos e atributos e valores textuais indexados. Apesar disso, existem outras informações mapeadas pela função *valor*, incluindo os identificadores de documentos e os códigos de eixo de atributo irmão que também são replicados entre vértices duplicados. A próxima subseção trata este problema através de um método que reorganiza as informações do índice a fim de reduzir ainda mais as informações de vértices duplicados.

#### 4.2.5 Tratamento da Replicação de Identificadores de Documentos e Códigos de Eixo

Em uma visão abstrata, um índice invertido é composto de uma lista de chaves de pesquisa em que cada chave aponta para um respectivo resultado de pesquisa. No índice gerado pela técnica BranchGuide, as chaves são os caminhos extraídos da fonte de dados. Já um resultado de pesquisa é composto por uma lista de identificadores de documentos nos quais ocorre um determinado caminho, e, além disso, para cada um dos identificadores de documento é armazenada uma lista de códigos de eixo de atributo irmão que são empregadas na recuperação de documentos com processamento de operadores lógicos. Por exemplo, a Figura 35 ilustra a organização de um índice invertido em que um conjunto de *caminhos*  $1...n$  é indexado. Um *i-ésimo* caminho indexado aponta para um *i-ésimo* resultado contendo as informações relacionadas com a ocorrência do dado caminho.

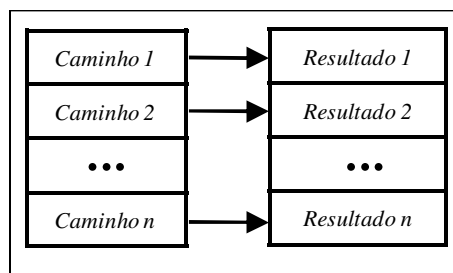


**Figura 35: Índice invertido de caminhos.**

Temos que dois diferentes vértices no grafo de índice  $G'$  são *duplicados* se estiverem em mesma altura em  $G'$  e, além disso, mapearem informações idênticas. Quando existem vértices duplicados no grafo de índice, os resultados mapeados por vértices duplicados geram

resultados replicados no índice final. Por exemplo, a Figura 36a mostra um índice em que um conjunto de *caminhos A...G* é indexado. No índice, cada caminho aponta para um respectivo *resultado A...G*. No exemplo, temos que os resultados dos *caminhos A, D e F* são replicados (supõe-se que os *caminhos A, D e F* sejam associados a três vértices duplicados).

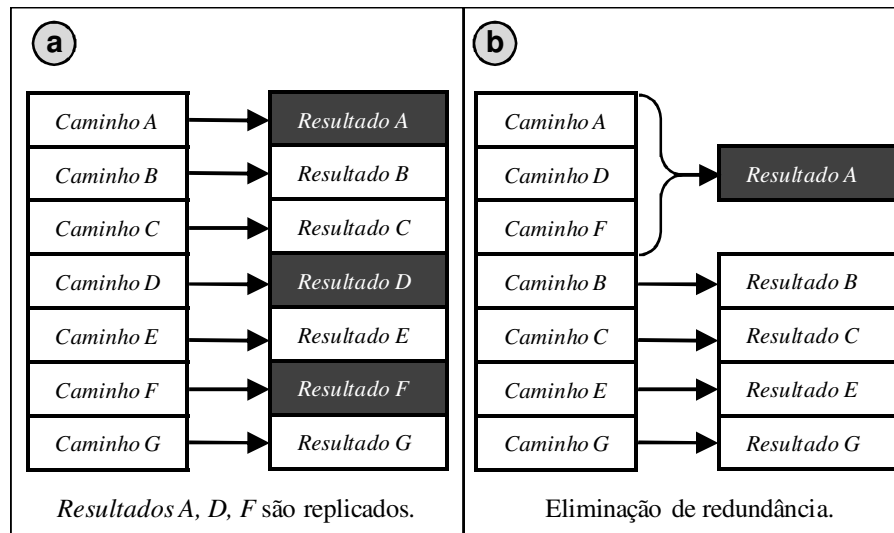


**Figura 36: Resultados replicados**

Neste contexto, a organização ideal de índice é aquela em que somente um dos resultados é armazenado e os três caminhos são agrupados de forma a apontarem para o mesmo resultado no índice e, assim, economizando as informações de ponteiros bem como reduzindo a redundância de dados. Esta organização ideal em que se elimina a redundância de dados é ilustrada na Figura 36b. Neste cenário, as chaves de caminhos com resultados duplicados são organizados no índice de forma sequencial formando um grupo que aponta para um único resultado não replicado. No exemplo da Figura 36b, as chaves dos *caminhos A, D e F* são armazenadas de forma sequencial, e os dados dos resultados *D, F* não precisam ser armazenados já que são idênticos aos dados do resultado *A*.

#### 4.2.5.1 Organização de Chaves de Índice

A fim de gerar uma organização de chaves de índice em que se reduz a duplicação de resultados, temos que ordenar os caminhos no índice de acordo com os seguintes critérios:

*i) Chaves de caminhos com vértice final de altura  $x$  surgem no índice antes de chaves de caminhos com vértice final de altura  $y$ , tal que  $x < y$ .*

*ii) Tomando o conjunto de caminhos com vértices finais de uma determinada altura  $h$ , para este conjunto de caminhos, as chaves de índice são ordenadas em função do código de*

*rótulo valor mais à direita até o código rótulo valor mais à esquerda pertencentes às sequências numéricas associadas a cada chave de índice.*

Por exemplo, tomando os vértices de altura 3 do grafo de índice ilustrado na Figura 33, pelos critérios anteriormente citados, as chaves de índice são ordenadas primeiramente levando em conta o último código das sequências numéricas, após isso, são ordenados em função do segundo código das sequências numéricas, e por fim são ordenados em função do primeiro código das sequências numéricas. A ordenação final é mostrada na Tabela 4, evidenciando o resultado esperado, assim os caminhos */editora[@nome="SparkNotes"]/livro* e */editora[@cidade="Oxford"]/livro* que estão associados a dois vértices duplicados são armazenados de forma sequencial no índice.

**Tabela 4: Ordenação de chaves de caminhos para a altura 3**

| Caminho                                   | Sequências Numéricas Ordenadas |
|-------------------------------------------|--------------------------------|
| <i>/editora[@nome="SparkNotes"]/livro</i> | 1, 1, 1                        |
| <i>/editora[@cidade="Oxford"]/livro</i>   | 1, 2, 1                        |
| <i>/editora/livro[@nome="Hamlet"]</i>     | 1, 3, 2                        |
| <i>/editora/livro[@nome="Othello"]</i>    | 1, 3, 3                        |

#### 4.2.5.2 Organização de Resultados de Índice

Dado o conjunto de resultados  $R = \{R_1 \dots R_n\}$ , onde  $n = \text{número total de chaves de índice}$  tal que o conjunto  $R$  está sequenciado de acordo com a ordenação das chaves de índice  $C = \{C_1 \dots C_n\}$ . Além disso, sendo que o resultado de pesquisa para a chave de índice  $C_i$  é  $R_i$ , então, temos que: *Caso uma entrada de resultado  $R_{i+1}$  ( $1 < i+1 \leq n$ ) seja replicação de uma entrada de resultado  $R_i$ , então o resultado  $R_{i+1}$  não é armazenado e o resultado de pesquisa para a chave  $C_{i+1}$  torna-se  $R_i$ .* Em outras palavras, de maneira menos formal, se tivermos duas chaves de índice sequenciais em que os resultados de pesquisa das duas chaves são replicados, então o resultado da segunda chave não é armazenado, e ao consultar pela segunda chave de índice o resultado de pesquisa é o mesmo da primeira chave de índice. Neste caso, define-se que a segunda chave **reusa** o resultado da primeira chave.

Por exemplo, na Figura 37, o Documento 3 é mostrado com os códigos de eixo de atributo irmão destacados. De posse de tais informações, no processo de indexação, os caminhos são extraídos do documento, e, além disso, para cada caminho extraído é indexado o identificador do documento no qual ocorre o caminho juntamente com as respectivas informações sobre códigos de eixo de atributo irmão.

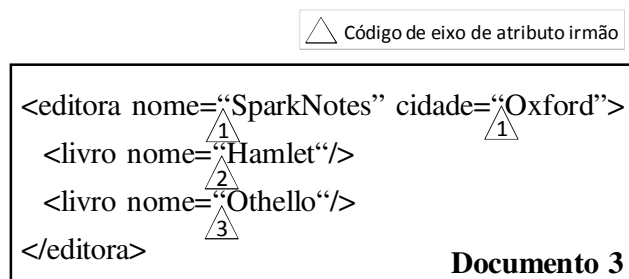


**Figura 37: Exemplo de documento com códigos de eixo de atributo irmão**

Após extração e indexação de caminhos, as chaves são ordenadas de acordo com a função de ordem definida na Seção 4.2.5.1. As chaves de índice ordenadas e o resultado de pesquisa para cada caminho extraído do *Documento 3* (Figura 37) são ilustrados na Tabela 5. Cada resultado é representado através da seguinte notação: *{Identificador de documento (Código de eixo de atributo irmão)}*; vale ressaltar que, caminhos que terminam em elementos não têm códigos de eixo de atributo irmão associados, logo, neste caso, o valor indexado é vazio ( $\emptyset$ ).

**Tabela 5: Indexação do Documento 3**

| Caminhos Ordenados                                    | Resultado                     | Resultado com Reuso           |
|-------------------------------------------------------|-------------------------------|-------------------------------|
| /editora                                              | {Documento 3 ( $\emptyset$ )} | {Documento 3 ( $\emptyset$ )} |
| /editora[ @nome="SparkNotes"]                         | {Documento 3 (1)}             | {Documento 3 (1)}             |
| /editora[ @cidade="Oxford"]                           | {Documento 3 (1)}             | ↑ Reuso                       |
| /editora/livro                                        | {Documento 3 ( $\emptyset$ )} | {Documento 3 ( $\emptyset$ )} |
| /editora[ @nome="SparkNotes"]/livro                   | {Documento 3 ( $\emptyset$ )} | ↑ Reuso                       |
| /editora[ @cidade="Oxford"]/livro                     | {Documento 3 ( $\emptyset$ )} | ↑ Reuso                       |
| /editora/livro[ @nome="Hamlet"]                       | {Documento 3 (2)}             | {Documento 3 (2)}             |
| /editora/livro[ @nome="Othello"]                      | {Documento 3 (3)}             | {Documento 3 (3)}             |
| /editora[ @nome="SparkNotes"]/livro[ @nome="Hamlet"]  | {Documento 3 (2)}             | {Documento 3 (2)}             |
| /editora[ @cidade="Oxford"]/livro[ @nome="Hamlet"]    | {Documento 3 (2)}             | ↑ Reuso                       |
| /editora[ @nome="SparkNotes"]/livro[ @nome="Othello"] | {Documento 3 (3)}             | {Documento 3 (3)}             |
| /editora[ @cidade="Oxford"]/livro[ @nome="Othello"]   | {Documento 3 (3)}             | ↑ Reuso                       |

Além disso, o tratamento dos resultados de índice replicados é aplicado sob o exemplo mostrado na Tabela 5, assim, eliminando resultados sequenciais replicados. Mais especificamente, na coluna *Resultado com Reuso* são listadas as entradas de resultado de pesquisa que são armazenadas ou reusadas após processo de indexação. As células desta coluna preenchidas com o símbolo ↑ **Reuso** indicam os resultados redundantes que não necessitam ser armazenados, mas ao invés disso, deve-se reusar o resultado anterior.

### 4.3 Considerações Finais

Na técnica BranchGuide, a indexação de caminhos com ramificação é um dos requisitos funcionais que impactam no aumento da quantidade de informação indexada e consequentemente no aumento do tamanho dos arquivos de índice. Dependendo da fonte de dados indexada, é possível ocorrer o reuso entre resultados de pesquisa por caminhos simples, mas, é necessário ressaltar que o algoritmo proposto é projetado principalmente para eliminar a replicação das informações geradas pelos caminhos com ramificação. Isto ocorre pelos seguintes fatores: *i)* O armazenamento de informações de caminhos simples contribui de forma linear para o crescimento do índice, logo, o tratamento da replicação dos resultados de pesquisa por tais caminhos tem contribuição pequena de redução de tamanho de índice. *ii)* Após aplicação da *Regra III* da técnica BranchGuide (Seção 3.2) é gerada uma grande quantidade de informações relacionadas aos caminhos de ramificação.

A aplicação da *Regra III* da técnica BranchGuide na geração do grafo de índice pode ser vista como uma expansão da quantidade de vértices que é no melhor caso linear (isto ocorre quando a fonte de dados não tem nenhum atributo). Ou seja, quanto maior a quantidade de atributos, maior o número de vértices recriados no grafo de índice e estes novos vértices recriados podem ser duplicados. Logo, o projeto de algoritmo para a técnica de BranchGuide minimiza os requisitos de armazenamento exigidos ao eliminar redundância de dados gerada na indexação de caminhos com ramificação. Por exemplo, na Tabela 6 estão destacados com sombreamento em escuro os caminhos com ramificação que tem resultados reusados após indexação do *Documento 3* (Figura 37).

**Tabela 6: Caminhos com ramificação reusados**

| <b>Caminhos Ordenados</b>                           | <b>Resultado</b>              |
|-----------------------------------------------------|-------------------------------|
| /editora                                            | {Documento 3 ( $\emptyset$ )} |
| /editora[@nome="SparkNotes"]                        | {Documento 3 (1)}             |
| /editora[@cidade="Oxford"]                          | ↑ Reuso                       |
| /editora/livro                                      | {Documento 3 ( $\emptyset$ )} |
| /editora[@nome="SparkNotes"]/livro                  | ↑ Reuso                       |
| /editora[@cidade="Oxford"]/livro                    | ↑ Reuso                       |
| /editora/livro[@nome="Hamlet"]                      | {Documento 3 (2)}             |
| /editora/livro[@nome="Othello"]                     | {Documento 3 (3)}             |
| /editora[@nome="SparkNotes"]/livro[@nome="Hamlet"]  | {Documento 3 (2)}             |
| /editora[@cidade="Oxford"]/livro[@nome="Hamlet"]    | ↑ Reuso                       |
| /editora[@nome="SparkNotes"]/livro[@nome="Othello"] | {Documento 3 (3)}             |
| /editora[@cidade="Oxford"]/livro[@nome="Othello"]   | ↑ Reuso                       |

Após indexação do *Documento 3*, os caminhos com ramificação duplicados */editora[@cidade="Oxford"]/livro* e *editora[@nome="SparkNotes"]/livro* não tem suas informações replicadas armazenadas no índice. Primeiramente, as informações sobre o valor textual do rótulo “livro” do vértice final do caminho */editora[@cidade="Oxford"]/livro* são armazenadas uma única vez através do tratamento de replicação de valores de rótulo e valores textuais do projeto de índice proposto (Seção 4.2.4). Em segundo, o valor de resultado de pesquisa ao consultar pelo caminho */editora[@cidade="Oxford"]/livro* é armazenado uma única vez através do tratamento de replicação de identificadores de documentos e códigos de eixo do projeto de índice proposto (Seção 4.2.5) através do reuso das informações do caminho */editora/livro*, conforme mostrado na Tabela 6.

O resultado obtido após indexação ao utilizar a técnica BranchGuide pode ser visto como um corte sob vértices duplicados do grafo de índice  $G'$ ; por exemplo, na Figura 38 é destacado um conjunto de vértices duplicados os quais foram gerados após aplicação da *Regra III* da técnica BranchGuide. Dentre estes vértices, alguns (destacados através de uma borda retangular pontilhada) não têm suas respectivas informações indexadas replicadas no índice, mas isto não implica diretamente que caso os caminhos associados a estes vértices sejam pesquisados os mesmos não serão encontrados. O corte citado é do ponto de vista de armazenamento e não do ponto de vista de redução de informações indexadas, desta forma, não causando perda de precisão.

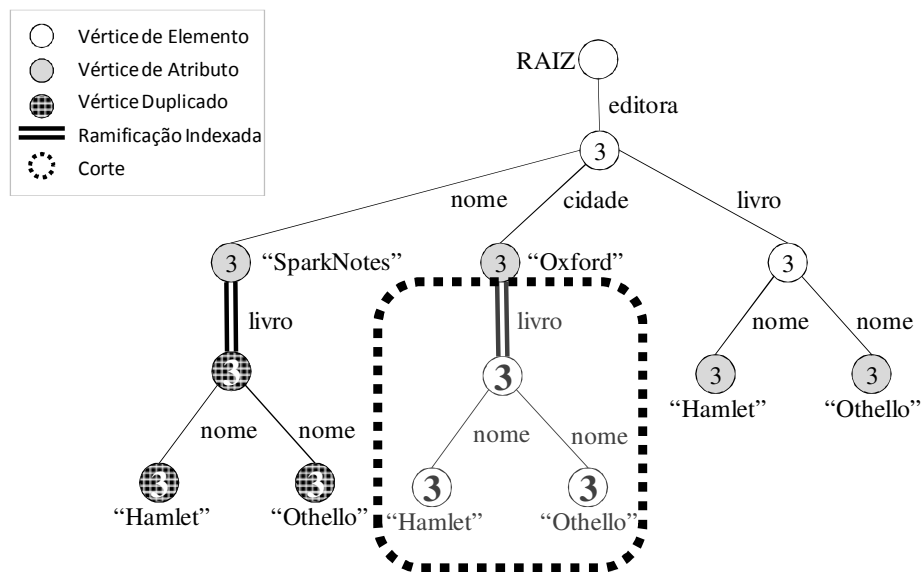


Figura 38: Corte de vértices duplicados

## 5 Implementação

A técnica de indexação BranchGuide adota a estrutura clássica de *índice invertido* para armazenamento dos índices, mapeando cada caminho para uma lista de resultados que contém identificadores de documentos. Como apontado por Manning [Manning08], o índice invertido é um mecanismo eficiente para recuperar uma lista de resultados para um determinado objeto pesquisado.

Especificamente, no processo de indexação, cada documento é carregado por vez e os caminhos a indexar são extraídos do documento de acordo com o padrão de seleção de caminhos definido na Seção 3.1. Estes caminhos são as chaves do índice invertido e para cada caminho indexado são armazenados como resultado de pesquisa as informações de ocorrência do caminho (ex: identificadores do documento), bem como, as informações auxiliares para posterior recuperação (ex: códigos de eixo de atributo irmão) definidas na Seção 4.1.1.

É importante ressaltar que na construção do índice, o uso de um algoritmo de ordenação é empregado para manter as chaves de índice em ordem, isto é essencial já que conforme novos documentos são indexados, caminhos (chaves de índice) previamente indexados devem ser localizados novamente e sua respectiva lista de resultados deve ser atualizada. Além disso, com as chaves ordenadas é possível empregar algoritmos de pesquisa mais eficientes do ponto de vista de complexidade de tempo de execução. Neste sentido, o algoritmo de ordenação de chaves empregado na implementação da técnica BranchGuide é detalhado e discutido neste capítulo, em específico, na Seção 5.1.

Também relacionado com a construção do índice é importante ressaltar que durante este processo, o índice é mantido em memória principal (RAM) e posteriormente gravado em memória secundária (disco rígido). Isto é feito somente se a quantidade de memória principal é suficiente para armazenar as informações do índice. Em contraste, em um cenário de aplicação real, a quantidade de documentos é geralmente ampla, de forma que uma larga coleção de documentos deve ser indexada. Neste caso em que a memória é insuficiente, é

necessário utilizar um método de *ordenação externa* [Manning08] (um método de ordenação que utilize memória secundária) para manter as chaves de índice ordenadas em memória secundária, e assim, viabilizar o processo de indexação. O processo de indexação bem como o algoritmo de ordenação externa adotados na implementação da técnica BranchGuide são detalhados na Seção 5.2 deste capítulo. Além disso, a forma dos arquivos gerados pelo processo de indexação é detalhada já que os arquivos têm um formato projetado bem como adotam técnicas de compressão. Também na Seção 5.2 o algoritmo de acesso aos arquivos de índice é detalhado, este algoritmo define como são recuperadas as informações indexadas.

## 5.1 Implementação do Método de Organização de Chaves de Índice

No arquivo de índice final é necessário que as chaves de índice e os resultados de pesquisa sejam armazenados de acordo com o projeto do algoritmo de indexação definido na Seção 4.2. Neste sentido, um algoritmo de ordenação é definido para manter a organização de chaves de índice de acordo com os critérios de organização definidos na Seção 4.2.5.1. A implementação do algoritmo de ordenação de chaves de índice é detalhada nesta seção.

Na implementação, as chaves do índice são armazenadas através de um código inteiro único para cada caminho indexado. A utilização de códigos inteiros possibilita utilizar técnicas de compressão (detalhadas na Seção 5.2.4) específicas para as chaves do índice invertido. A fim de atribuir códigos exclusivos para cada caminho é empregada uma função de *hash* perfeito  $h$  que mapeia sem colisões um dado caminho  $p$  codificado como sequência numérica para um único código inteiro  $h(p)$ . Portanto, para todo caminho  $p_1$  e  $p_2$ , então  $h(p_1) \neq h(p_2) \Leftrightarrow p_1 \neq p_2$ . A próxima subseção detalha a definição da função de *hash* perfeito adotada para conversão de sequências numéricas em códigos inteiros.

### 5.1.1 A Função de *hash* perfeito

Dado um conjunto de sequências numéricas  $N$  tal que o tamanho da maior sequência pertencente a  $N$  é  $T$ , então toda sequência em  $N$  é representada por uma tupla de  $T$  termos. Define-se que todo conjunto de sequências numéricas  $N$  é mapeado para de um conjunto  $M$  de tuplas de  $T$  termos. Neste mapeamento, para uma sequência numérica  $s \in N$  com tamanho  $t$  menor do  $T$ , temos que a tupla representante em  $M$  é dada por uma tupla tal que os primeiros  $t$  números da sequência correspondem aos  $t$  primeiros termos, e a partir do  $(t+1)$ -ésimo termo até o  $T$ -ésimo termo, os valores de tupla são definidos com o número zero. Por exemplo, supondo que o maior tamanho de sequência é 5, então, para a sequência numérica 2, 2, 1 a tupla representante é (2, 2, 1, 0, 0). Já para uma sequência  $s \in N$  com tamanho  $T$ , temos que a

tupla representante em  $M$  é dada por uma tupla tal que os  $T$  primeiros números da sequência numérica correspondem aos  $T$  primeiros termos. Por exemplo, supondo que o maior tamanho de sequência é 5, então, para a sequência numérica 2, 2, 1, 3, 4 a tupla representante é (2, 2, 1, 3, 4).

Para fins de exemplificação do uso da função de *hash* perfeito  $h$  é utilizado um conjunto de sequências numéricas listadas na Tabela 7. Neste caso, temos que o conjunto de tuplas representantes para tal conjunto de sequências numéricas também é listado na Tabela 7. Vale ressaltar que neste exemplo,  $T = 3$ .

**Tabela 7: Conjunto de tuplas representantes**

| Sequência Numérica $\in N$ | Tupla Representante $\in M$ |
|----------------------------|-----------------------------|
| 0                          | (0, 0, 0)                   |
| 1, 1, 1                    | (1, 1, 1)                   |
| 2, 3                       | (2, 3, 0)                   |
| 1, 2                       | (1, 2, 0)                   |

A função de *hash* perfeito  $h$  é dependente de dois conjuntos numéricos extraídos a partir do conjunto de tuplas representantes  $M$ :

i) O primeiro é o *conjunto de quantidade de distintos números por ordem* =  $Q$ . O  $i$ -ésimo ( $1 \leq i \leq T$ ) termo de  $Q$  é dado pela quantidade de distintos números que ocorrem em todos os  $i$ -ésimos termos de todas as tuplas de  $M$ . Por exemplo, para os primeiros termos das tuplas do exemplo da Tabela 7 ocorrem os três distintos números  $\{0, 1, 2\}$ , logo  $Q_1 = 3$ . Já para os segundos termos das tuplas do exemplo da Tabela 7 ocorrem os quatro distintos números  $\{0, 1, 2, 3\}$ , logo  $Q_2 = 4$ . Por fim, para os terceiros termos das tuplas do exemplo da Tabela 7 ocorrem os dois distintos números  $\{0, 1\}$ , logo  $Q_3 = 2$ . Portanto,  $Q = \{Q_1, Q_2, Q_3\} = \{3, 4, 2\}$ .

ii) O segundo é o *conjunto de base* =  $B$ . Este conjunto é diretamente empregado pela função de *hash* perfeito  $h$  para mapear uma sequência numérica para um código inteiro. Os termos de  $B$  são definidos em função do conjunto  $Q$  anteriormente citado. Formalmente, os termos de  $B$  são definidos de acordo com a equação mostrada na Figura 39.

$$B = \begin{cases} B_T = 1, \\ B_i = \prod_{j=\{(i+1) \dots T\}} Q_j, \text{ para } 1 \leq i < T \end{cases}$$

**Figura 39: Equação do conjunto de base**

De acordo com a equação mostrada na Figura 39, o  $T$ -ésimo termo de  $B$  é 1, já o  $i$ -ésimo ( $1 \leq i < T$ ) termo de  $B$  é dado pelo produto do  $(i + 1)$ -ésimo termo de  $Q$  até o  $T$ -ésimo termo de  $Q$ . Para o exemplo em que  $Q = \{3, 4, 2\}$  e  $T = 3$ , temos que o  $B_3 = 1$ . Por sua vez,  $B_1 = 4 * 2 = 8$ , e por fim,  $B_2 = 2$ . Portanto,  $B = \{8, 2, 1\}$ .

De posse do conjunto de base  $B$ , define-se que para uma sequência numérica de tamanho  $T$  tal que a tupla representante é  $P$ , então  $h(P)$  é definida pela equação dependente do conjunto de base  $B$  mostrada na Figura 40. O código inteiro  $h(P)$  é dado pelo somatório do produto do  $i$ -ésimo termo de  $P$  pelo  $i$ -ésimo termo do conjunto de base  $B$  tal que ( $1 \leq i \leq T$ ). Para o exemplo mostrado na Tabela 7, temos um conjunto de base  $B = \{8, 2, 1\}$ ; logo, através da função de *hash* perfeito  $h$ , a sequência numérica 2, 3, 0 que tem a tupla representante definida por  $P = (2, 3, 0)$  é mapeada para o valor de código inteiro  $h(P)$  dado por  $h(P) = 2*8 + 3*2 + 0*1 = 22$ .

$$h(P) = \sum_{j=(1 \dots T)} P_j * B_j$$

**Figura 40: Função de hash perfeito  $h$**

Para evidenciar que a função de *hash* empregada é perfeita, isto é,  $h(P)$  é uma função que mapeia distintas tuplas para distintos códigos inteiros sem ocorrência de colisão; o conjunto de tuplas de tamanho  $T = 3$  tal que para o primeiro termo podem ocorrer os números  $\{0, 1, 2\}$ , para o segundo termo podem ocorrer os números  $\{0, 1, 2, 3\}$ , e por fim, para o terceiro termo podem ocorrer os números  $\{0, 1\}$  é mapeado para códigos inteiros sem colisão conforme mostrado na Tabela 8. Vale ressaltar que, o valor de conjunto de base deste exemplo é o mesmo do exemplo mostrado anteriormente na Tabela 7.

**Tabela 8: Mapeamento perfeito**

| $P$       | $h(P)$ | $P$       | $h(P)$ |
|-----------|--------|-----------|--------|
| (0, 0, 0) | 0      | (1, 2, 0) | 12     |
| (0, 0, 1) | 1      | (1, 2, 1) | 13     |
| (0, 1, 0) | 2      | (1, 3, 0) | 14     |
| (0, 1, 1) | 3      | (1, 3, 1) | 15     |
| (0, 2, 0) | 4      | (2, 0, 0) | 16     |
| (0, 2, 1) | 5      | (2, 0, 1) | 17     |
| (0, 3, 0) | 6      | (2, 1, 0) | 18     |
| (0, 3, 1) | 7      | (2, 1, 1) | 19     |
| (1, 0, 0) | 8      | (2, 2, 0) | 20     |
| (1, 0, 1) | 9      | (2, 2, 1) | 21     |
| (1, 1, 0) | 10     | (2, 3, 0) | 22     |
| (1, 1, 1) | 11     | (2, 3, 1) | 23     |

A função de *hash* perfeito  $h$  é empregada na ordenação de caminhos no processo de indexação, já que para cada sequência numérica associada a um dado caminho é gerado um exclusivo código inteiro, então, os caminhos podem ser ordenados de acordo com a ordenação usual de valores inteiros. O algoritmo de ordenação de caminhos (chaves de índice) é detalhado na próxima subseção.

### 5.1.2 Ordenação de Chaves de Índice

A implementação do processo de ordenação de chaves de índice considera as premissas de organização de chaves de índice definidas na Seção 4.2.5.1. Neste sentido, de acordo com método de organização de chaves proposto são definidos os seguintes requisitos para o algoritmo de ordenação de chaves:

*i) Como resultado da ordenação, toda chave de índice que tem sequência numérica associada de tamanho  $x$  surge antes de toda chave de índice que tem sequência numérica associada de tamanho  $y$  tal que  $x < y$ ;*

*ii) Além disso, tomando o conjunto de todos os caminhos associados à sequências numéricas de um mesmo tamanho, para este conjunto de caminhos, as chaves de índice são ordenadas em função do código de rótulo-valor mais à direita até o código de rótulo-valor mais à esquerda pertencentes às sequências numéricas de cada chave de índice.*

Por exemplo, de acordo com os requisitos citados, após o processo de ordenação, os caminhos (destacados em negrito na Tabela 9)  $p_1 = /editora[@nome="SparkNotes"]/livro$  e  $p_2 = /editora[@cidade="Oxford"]/livro$  devem ser armazenados sequencialmente pois:

**Tabela 9: Caminhos codificados e tuplas correspondentes**

| Caminho                                             | Sequência Numérica | Tupla Representante |
|-----------------------------------------------------|--------------------|---------------------|
| /editora                                            | 1                  | (1, 0, 0, 0)        |
| /editora[@nome="SparkNotes"]                        | 1, 1               | (1, 1, 0, 0)        |
| <b>/editora[@nome="SparkNotes"]/livro</b>           | <b>1, 1, 1</b>     | (1, 1, 1, 0)        |
| /editora[@nome="SparkNotes"]/livro[@nome="Hamlet"]  | 1, 1, 1, 1         | (1, 1, 1, 1)        |
| /editora[@nome="SparkNotes"]/livro[@nome="Othello"] | 1, 1, 1, 2         | (1, 1, 1, 2)        |
| /editora[@cidade="Oxford"]                          | 1, 2               | (1, 2, 0, 0)        |
| <b>/editora[@cidade="Oxford"]/livro</b>             | <b>1, 2, 1</b>     | (1, 2, 1, 0)        |
| /editora[@cidade="Oxford"]/livro[@nome="Hamlet"]    | 1, 2, 1, 1         | (1, 2, 1, 1)        |
| /editora[@cidade="Oxford"]/livro[@nome="Othello"]   | 1, 2, 1, 2         | (1, 2, 1, 2)        |
| /editora/livro                                      | 1, 3               | (1, 3, 0, 0)        |
| /editora/livro[@nome="Hamlet"]                      | 1, 3, 2            | (1, 3, 2, 0)        |
| /editora/livro[@nome="Othello"]                     | 1, 3, 3            | (1, 3, 3, 0)        |

i) O caminho  $p_1$  é associado à sequência “1, 1, 1” de tamanho igual a 3 e o caminho  $p_2$  é associado à sequência “1, 2, 1” de tamanho também igual a 3.

ii) Além disso, as sequências “1, 1, 1” e “1, 2, 1” devem ser ordenadas levando em consideração primeiramente o código rótulo valor mais a direita até o código rótulo valor mais à esquerda, assim, o código “1, 1, 1” deve surgir antes do código “1, 2, 1” após resultado final de ordenação.

Intuitivamente, uma maneira para que os caminhos  $/editora[@nome=“SparkNotes”]/livro$  e  $/editora[@cidade=“Oxford”]/livro$  sejam dispostos em sequência é ordená-los com prioridade sob os vértices finais do caminhos, e não sob os vértices iniciais. Em outras palavras, os caminhos devem ser ordenados com prioridade sob os códigos rótulo-valor finais (mais à direita) das sequências associadas e não sob os códigos iniciais (mais à esquerda). Assim, seriam agrupados os caminhos que têm os vértices que terminam no elemento *livro*, e não os caminhos que iniciam a partir do elemento *editora*.

Para obter este resultado, invertem-se os caminhos a serem ordenados em caminhos hipotéticos  $livro/[@nome=“SparkNotes”]editora/$  e  $livro/[@cidade=“Oxford”]editora/$  e, com isto, é aplicado sob tais caminhos hipotéticos a função de *hash* perfeito  $h$  para assim obter um conjunto de códigos inteiro. A ordenação decimal deste conjunto de códigos inteiros gera um conjunto de caminhos ordenados de acordo com os requisitos de ordenação especificados. Especificamente, o algoritmo de ordenação é composto das seguintes etapas:

i) *Aplica-se uma inversão em cada tupla representante  $P_i \in M$  tal que  $1 \leq i \leq \text{quantidade de caminho indexados}$ ; a inversão consiste em gerar um conjunto de tuplas  $M^{-1}$  tal que a ordem dos termos de cada tupla  $P_i^{-1} \in M^{-1}$  é contrária à ordem dos termos da respectiva tupla  $P_i \in M$ .*

ii) *Aplica-se a função de hash perfeito  $h$  separadamente sob cada tupla  $P_i^{-1} \in M^{-1}$  gerando assim um conjunto de códigos inteiros  $C_i \in C$ .*

iii) *O conjunto de códigos inteiros  $C$  é ordenado através de ordenação decimal.*

Tomando como exemplo os caminhos e respectivo conjunto  $M$  de tuplas representantes listadas na Tabela 9, temos que cada tupla representante  $P_i \in M$  é invertida para uma tupla  $P_i^{-1} \in M^{-1}$  e a função de *hash* perfeito  $h$  é aplicada sob cada tupla representante invertida  $P_i^{-1}$ , desta forma, gerando os respectivos códigos inteiros para cada caminho indexado. O resultado obtido é mostrado na Tabela 10.

**Tabela 10: Inversão de tuplas e aplicação da função de hash**

| Tupla Representante<br>$P_i \in M$ | Tupla Representante<br>Invertida $P_i^{-1} \in M^I$ | Código Inteiro<br>$C_i = h(P_i^{-1}) \in C$ |
|------------------------------------|-----------------------------------------------------|---------------------------------------------|
| (1, 0, 0, 0)                       | (0, 0, 0, 1)                                        | 1                                           |
| (1, 1, 0, 0)                       | (0, 0, 1, 1)                                        | 3                                           |
| (1, 1, 1, 0)                       | (0, 1, 1, 1)                                        | 11                                          |
| (1, 1, 1, 1)                       | (1, 1, 1, 1)                                        | 43                                          |
| (1, 1, 1, 2)                       | (2, 1, 1, 1)                                        | 74                                          |
| (1, 2, 0, 0)                       | (0, 0, 2, 1)                                        | 5                                           |
| (1, 2, 1, 0)                       | (0, 1, 2, 1)                                        | 13                                          |
| (1, 2, 1, 1)                       | (1, 1, 2, 1)                                        | 45                                          |
| (1, 2, 1, 2)                       | (2, 1, 2, 1)                                        | 77                                          |
| (1, 3, 0, 0)                       | (0, 0, 3, 1)                                        | 7                                           |
| (1, 3, 2, 0)                       | (0, 2, 3, 1)                                        | 22                                          |
| (1, 3, 3, 0)                       | (0, 3, 3, 1)                                        | 31                                          |

Após ordenação decimal do conjunto  $C$  de códigos inteiros é obtido o resultado de ordenação final dos caminhos já que cada código  $C_i \in C$  é associado a um determinado caminho indexado. O resultado final da ordenação dos caminhos é mostrado na Tabela 11. Vale ressaltar que, a ordem dos caminhos está de acordo com as restrições de ordenação requeridas e, conseqüentemente, de acordo as regras de organização de chaves do índice inicialmente apresentadas na Seção 4.2.5.1.

**Tabela 11: Resultado final da ordenação**

| Código Inteiro | Caminhos Ordenados                                    |
|----------------|-------------------------------------------------------|
| 1              | /editora                                              |
| 3              | /editora[ @nome="SparkNotes"]                         |
| 5              | /editora[ @cidade="Oxford"]                           |
| 7              | /editora/livro                                        |
| 11             | /editora[ @nome="SparkNotes"]/livro                   |
| 13             | /editora[ @cidade="Oxford"]/livro                     |
| 22             | /editora/livro[ @nome= "Hamlet"]                      |
| 31             | /editora/livro[ @nome= "Othello"]                     |
| 43             | /editora[ @nome="SparkNotes"]/livro[ @nome="Hamlet"]  |
| 45             | /editora[ @cidade="Oxford"]/livro[ @nome="Hamlet"]    |
| 74             | /editora[ @nome="SparkNotes"]/livro[ @nome="Othello"] |
| 77             | /editora[ @cidade="Oxford"]/livro[ @nome="Othello"]   |

Por fim, é importante ressaltar que a inversão de tuplas e aplicação da função de *hash* perfeito  $h$  também são executadas no processo de consulta, mas, neste caso, o algoritmo é

aplicado sob uma única tupla e o código inteiro é localizado no índice. Tais questões são detalhadas na próxima seção.

## 5.2 Implementação do Índice

Nesta seção, as questões de implementação do índice gerado com a técnica BranchGuide são discutidas. Inicialmente, é importante ressaltar que o *Índice* é um componente de um *Sistema de Busca*, conforme apresentado na Seção 1.3. Além do Índice, no sistema de busca existem dois componentes denominados como *Motor de Indexação* e *Motor de Busca* que são respectivamente responsáveis por gerar o Índice, e assim, acessá-lo para efetuar pesquisas. Nas próximas subseções, a arquitetura e organização dos dados do Índice são descritas. Além disso, os processos de indexação e de consulta do Índice são especificados. Por fim, os métodos de compressão aplicados no armazenamento do Índice são discutidos.

Em uma visão abstrata, o *Índice* (Figura 41c) é composto de dois índices invertidos. O primeiro índice, denominado *Tabela de Mapeamento de Códigos Rótulo-Valor* (Figura 41a), mapeia assinaturas constituídas de pares (*rótulo*, *valor*) (conforme definido na Seção 4.2.2) para valores numéricos. O segundo índice, denominado de *Índice de Caminhos* (Figura 41b) mapeia cada caminho indexado para uma lista de resultados de pesquisa.

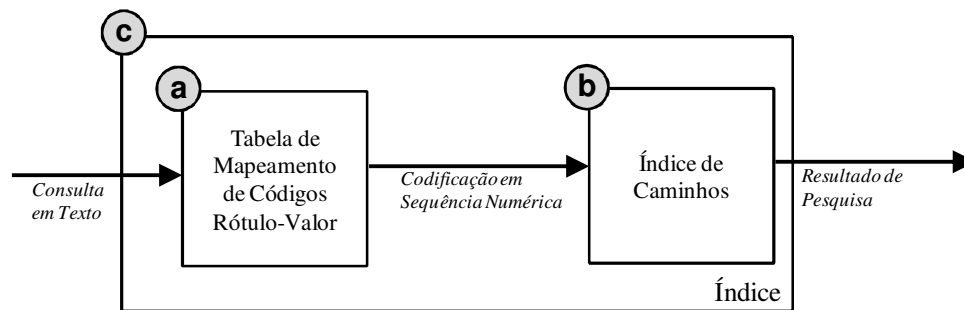


**Figura 41: Arquitetura do índice**

Neste sentido, um processo de indexação deve gerar um Índice com as características definidas pela arquitetura apresentada na Figura 41 a partir de uma fonte de dados de documentos. O processo de indexação adotado na geração de Índice empregando a técnica BranchGuide é detalhado na seção seguinte.

### 5.2.1 Processo de Indexação

O fluxograma mostrado na Figura 42 especifica o processo de indexação da técnica BranchGuide. Neste processo, o objetivo principal é gerar como saída um arquivo de índice tendo como entrada os documentos armazenados na fonte de dados. Mais especificamente, as atividades mostradas no fluxograma são desempenhadas pelo *Motor de Indexação*, que é

responsável por indexar uma *Fonte de Dados* empregando a técnica BranchGuide, e assim, gerar o *Índice*. É importante ressaltar que o Motor de Indexação, o Índice bem como a Fonte de Dados são componentes que fazem parte de um *Sistema de Busca* (Figura 1), a fim de prover as funcionalidades de recuperação e indexação.

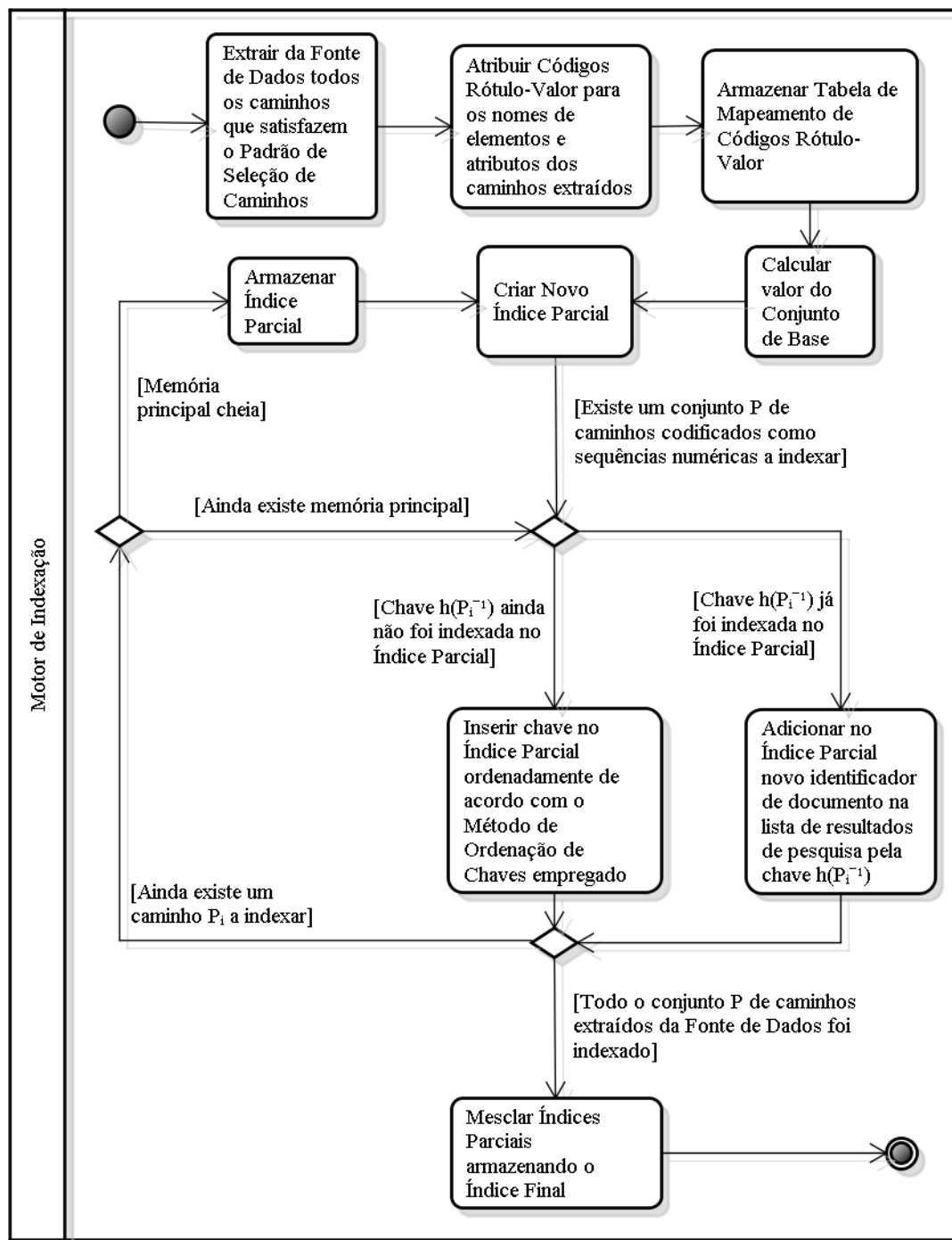


Figura 42: Processo de indexação

Inicialmente, no processo de indexação, temos que todos os caminhos que conformam com o padrão de seleção de caminhos suportado (Seção 3.1) pela técnica BranchGuide são extraídos da Fonte de Dados. Cada caminho extraído é decomposto em nomes de elementos/atributos de forma que para cada nome existente em cada nível é mapeado um código de rótulo-valor seguindo o método de mapeamento de códigos rótulo-valor discutido na Seção 4.2.2. O mapeamento final é armazenado na Tabela de Mapeamento de Códigos. Rótulo-Valor (Figura 41a).

Além disso, após o mapeamento, é possível determinar o tamanho do maior caminho existente bem como a quantidade de códigos distintos por nível. Estas informações são necessárias para gerar a computação do conjunto de base de acordo com o procedimento discutido na Seção 5.1. O conjunto de base é uma informação essencial, já que é utilizado no processamento da conversão de caminhos em sequências de códigos numéricos. De posse do mapeamento e do conjunto de base o processo de construção do índice pode ser iniciado.

Primeiramente, é importante ressaltar que durante o processo de construção do índice toda vez que uma nova chave de índice é armazenada, esta nova chave deve ser inserida ordenadamente. Adotar um algoritmo para ordenação de chaves permite localizar a chave em um pós-momento de pesquisa. Os algoritmos de ordenação tradicionais como o *quick sort* e o *merge sort* [Cormen01] apresentam desempenho não adequado quando utilizados em dados armazenados em disco rígido, pois, de fato, unidades de discos rígidos são extremamente lentas para localizar entradas em posições físicas arbitrárias (isto é, via acesso aleatório). Ao contrário, a leitura de dados sequencial é relativamente rápida em comparação com o acesso aleatório, de forma que é o modo mais rápido para ler dados a partir de discos rígidos [Manning08]. Por exemplo, o algoritmo *external multi-way merge sort* [Saltenis04] possibilita ordenar dados em discos rígidos utilizando leitura sequencial.

Em linhas gerais, o *external multi-way merge sort* ordena em memória de acesso aleatório (RAM) a maior quantidade de dados possível, utilizando para isto um algoritmo de ordenação eficiente, tal como o *merge sort*. Após os dados serem ordenados em memória, estes são armazenados em disco em um arquivo parcial de indexação (*Índice Parcial*). Este procedimento é repetido até que todos os dados da fonte de dados tenham sido ordenados por este processo. Por fim, uma etapa de mesclagem final é executada sob os arquivos de indexação parciais, gerando um único arquivo de indexação (*Índice Final*).

O algoritmo *external multi-way merge sort* é empregado no processo de indexação. Neste sentido, inicialmente é criado um Índice Parcial (em memória de acesso aleatório). Com

isto, para cada novo caminho extraído de um documento, caso este caminho não esteja indexado no Índice Parcial então uma nova entrada de índice é criada. Vale ressaltar que o caminho em formato textual é convertido para uma sequência numérica através da Tabela de Mapeamento de Código Rótulo-Valor e, com isto, a função de *hash* perfeito é calculada sob o a inversão desta sequência numérica obtendo-se um código decimal que identifica o caminho indexado no índice (conforme discutido na Seção 5.1).

Caso o caminho já tenha sido anteriormente adicionado no Índice Parcial, então não é necessário adicionar novamente a nova chave de índice, ao invés disso, o resultado de pesquisa associado com a chave de índice é atualizado ao adicionar uma nova entrada relativa ao documento em que ocorre o caminho.

Durante a indexação de um novo caminho, caso o Índice Parcial não caiba mais em memória principal, então este é armazenado temporariamente em memória secundária, a partir disto, o processo de indexação segue empregando um novo Índice Parcial. Este procedimento é repetido até que todos os caminhos da fonte de dados tenham sido indexados. Por fim, a etapa de mesclagem de índices parciais é ativada, assim, gerando o arquivo de Índice Final em memória secundária.

É importante ressaltar que o armazenamento final das chaves leva em conta a ordenação decimal dos códigos inteiros computados utilizando a função de *hash* perfeito adotada na implementação. Pelo fato de empregar esta função de *hash* perfeito, temos que a ordenação resultante das chaves de índice conforma com os requisitos de organização das chaves de índice apresentados na Seção 4.2.5.1. Além disso, no armazenamento do Índice Final, temos que os resultados de pesquisa duplicados não são armazenados, como de acordo com os requisitos de organização de resultados de pesquisa apresentados na Seção 4.2.5.2.

### **5.2.2 Processo de Consulta ao Índice**

Enquanto que um processo de indexação é responsável por gerar um índice a partir de uma fonte de dados, temos que o processo de consulta é responsável por ler o índice gerado e recuperar os documentos que satisfazem restrições expressas através de consultas. Dentro do Sistema de Busca (Figura 1), o componente do Motor de Busca tem a função de executar as ações para o processamento de consultas. Nesta seção, as atividades executadas pelo Motor de Busca para processar consultas adotando a técnica BranchGuide são detalhadas. Assim, o fluxograma mostrado na Figura 43 especifica as atividades executadas no processamento de consultas.

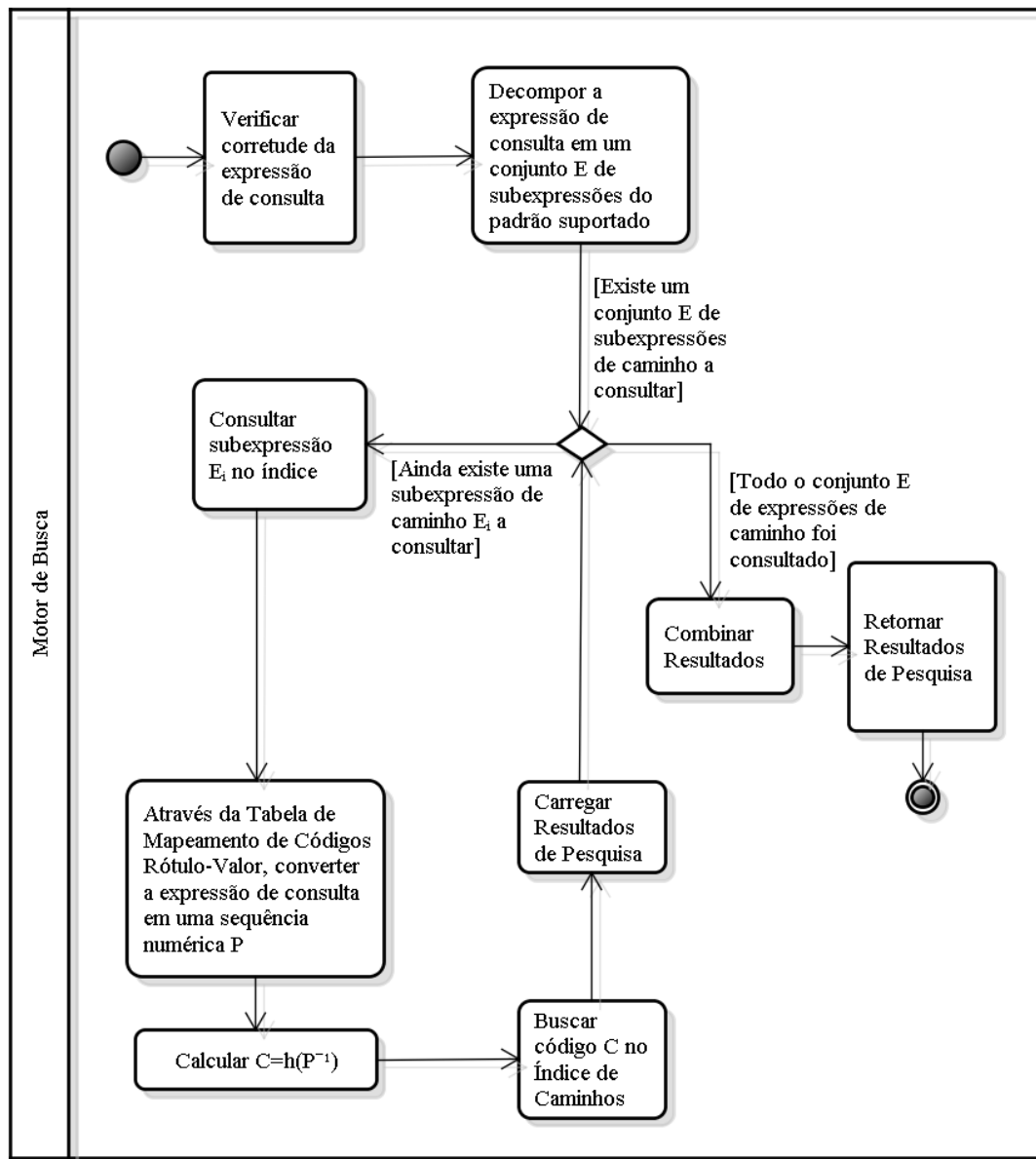


**Figura 43: Processamento de uma consulta**

Inicialmente é verificado se existem erros de sintaxe na expressão de consulta. Caso não existam erros, o processamento da consulta é iniciado.

Verificada positivamente a sintaxe da expressão, existem dois casos em que a expressão de consulta pode ser classificada. No caso mais simples, a expressão de consulta satisfaz o padrão de seleção de caminhos (Seção 3.1) adotado pela técnica BranchGuide, e assim, não contém operadores de controle de similaridade (Seção 3.3). Em contrapartida, no caso mais complexo, a expressão de consulta contém operadores de controle de similaridade e deve ser decomposta em um conjunto de subexpressões de consultas que satisfazem o padrão de seleção de consultas.

No caso mais simples, isto é, quando não existem operadores de similaridade, o conjunto de subexpressões a processar contém somente uma única expressão que satisfaz o padrão de seleção de consultas. No caso mais complexo, a quantidade de subexpressões a processar aumenta conforme é maior a quantidade de operadores de similaridade na expressão de consulta e, assim, é gerado um conjunto com mais do que uma subexpressão a processar.

Se a expressão de consulta contém operadores de similaridade, então deve ser decomposta em um conjunto de subexpressões que conformam com o padrão de seleção suportado pela técnica BranchGuide. Cada subexpressão é pesquisada no Índice separadamente. Especificamente, as subexpressões decompostas (que não contém operadores de similaridade) são pesquisadas no Índice de acordo com os seguintes passos:

Primeiramente, o Índice recebe como entrada uma expressão de consulta expressa em forma textual, desta forma, a expressão é codificada para uma sequência numérica que é a entrada para o Índice de Caminhos (Figura 41b). A codificação é executada pela Tabela de Mapeamento de Códigos Rótulo-Valor (Figura 41a). Esta tabela mantém o mapeamento (*assinatura, nível*)  $\rightarrow$  *código de rótulo-valor* (Seção 4.2.2), de forma que uma expressão de consulta pode ser convertida em uma sequência numérica composta por códigos de rótulo-valor. É importante ressaltar que esta tabela mapeia informação textual para informação numérica, de forma que é implementada como um índice invertido. A implementação da Tabela de Mapeamento de Códigos Rótulo-Valor utiliza as diretivas padrão para projeto de índices invertido de texto definidas por [Manning08].

Já o Índice de Caminhos recebe como entrada um caminho codificado como sequência numérica e sob esta sequência gera uma tupla representante  $P$ . Para gerar a tupla representante o Índice de Caminhos mantém armazenado o tamanho  $T$  da maior sequência numérica indexada. Após isto, a tupla  $P$  é invertida para a tupla  $P^{-1}$ , e por fim,  $P^{-1}$  é mapeada para o código inteiro  $C = h(P^{-1})$ . O código inteiro  $C$  é localizado no índice a fim de recuperar o respectivo resultado de pesquisa. É importante ressaltar que o conjunto de base  $B$  é armazenado no Índice de Caminhos para viabilizar o cálculo da função de *hash*  $h$  em tempo de execução da consulta.

O número de vezes em que Índice é acessado é igual ao número de subexpressões decompostas. Após os resultados serem recuperados, estes são combinados a fim de gerar o resultado final do processamento da expressão de consulta com operadores de controle de similaridade. O processo de combinação de resultados segue os procedimentos detalhados na Seção 4.1.

### 5.2.3 Organização do Índice de Caminhos

Apesar de não ser um índice de termos textuais (mas de códigos inteiros), o Índice de Caminhos é organizado utilizando diretivas baseada nas recomendações de projeto de índices de texto definidas por [Manning08]. Nesta subseção, a organização do Índice de Caminhos é detalhada e discutida.

Cada caminho indexado gera o armazenamento de um par  $(C, p_i)$  em que  $C$  é o código inteiro do respectivo caminho e o  $p_i$  é um endereço para o arquivo de resultado de pesquisa; a partir deste ponteiro é possível carregar os resultados de pesquisa. Para possibilitar uma busca eficiente, adota-se uma estratégia de indexação em *blocos*. Um bloco é um conjunto sequencial de  $k$  pares  $(C, p_i)$  armazenados (Figura 44b).

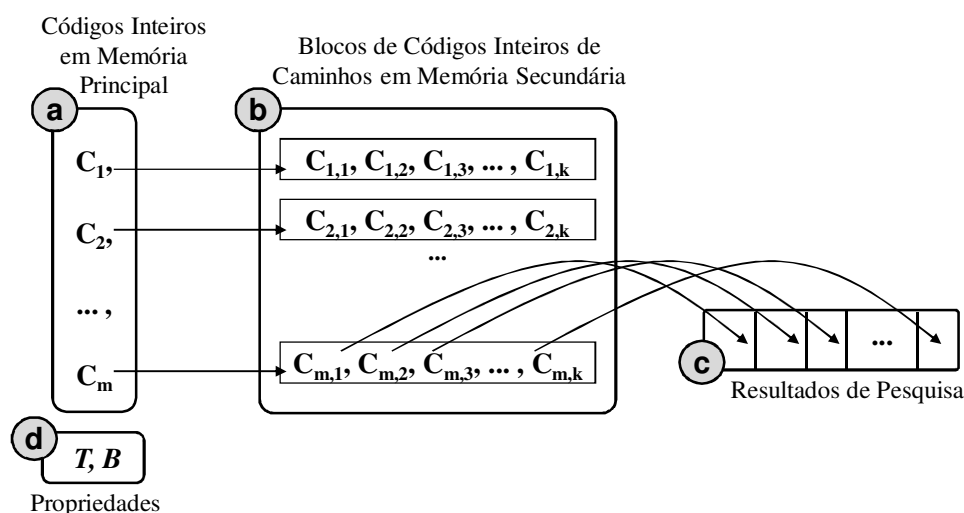


**Figura 44: Organização do índice de caminhos**

Adotando esta estratégia, os primeiros códigos inteiros de cada bloco são mantidos ordenados em memória principal (Figura 44a); juntamente com os endereços para os blocos em memória secundária no qual cada código inteiro pertence. Ao executar uma pesquisa, o código inteiro  $C$  é pesquisado em memória principal utilizando uma busca binária. A busca binária retorna o endereço para bloco em memória secundária onde o código  $C$  pode estar armazenado. Em seguida, o bloco é carregado em memória principal e o código  $C$  é pesquisado sequencialmente neste bloco. Se o par  $(C, p_i)$  for encontrado, então o resultado de pesquisa (Figura 44c) apontado pelo endereço  $p_i$  é carregado e retornado pelo índice.

É importante ressaltar que quanto mais pares  $(C, p_i)$  são armazenados em cada bloco, mais lenta é a recuperação de informações pois o tempo de pesquisa linear no bloco tende a ser maior. Experimentos preliminares evidenciaram que para  $k > 128$  o tempo de recuperação

começa a crescer. Neste sentido, no índice implementado, o valor de  $k$  é definido como 128, isto é, são armazenados 128 pares  $(C, p_i)$  por bloco.

Além disso, não somente os primeiros códigos inteiros de cada bloco são carregados e mantidos em memória. Temos que as propriedades do índice (Figura 44d) que incluem o maior tamanho  $T$  de tupla indexada e o conjunto de base  $B$  são carregados e mantidos em memória após inicialização do índice. Estas informações são utilizadas para computar códigos inteiros a partir de sequências numéricas utilizando a função de *hash* perfeito  $h$ . Já que estas informações auxiliares ficam carregadas em memória, o tempo de computação da função de *hash*  $h$  não degrada o desempenho do processamento de expressões de consulta.

Por fim, vale ressaltar que os dados do índice são armazenados utilizando técnicas de compressão detalhadas na próxima seção.

#### 5.2.4 Compressão de Índice

Como apontado por Manning [Manning08], o emprego de técnicas de compressão para informações do índice é essencial. Um benefício óbvio do uso de técnicas de compressão é que menos espaço em disco é utilizado. Além disso, através de técnicas de compressão, mais informação pode ser carregada de uma única vez do disco para a memória principal. Desta forma, como o número de operações de acesso a disco ao carregar informações é menor, obtém-se uma melhoria substancial do desempenho dos sistemas de recuperação.

Na implementação da técnica BranchGuide, técnicas de compressão são empregadas em diferentes informações armazenadas no índice. É importante ressaltar que as técnicas de descompressão empregadas são altamente eficientes para não degradar o desempenho do sistema de recuperação. Além disso, todas as técnicas de compressão empregadas não causam perda de dados, pois toda informação descomprimida é idêntica à informação original. A seguir, as técnicas de compressão empregadas são descritas.

*i)* A técnica de *compressão por prefixo* [Manning08] explora a ordenação lexicográfica de termos textuais cujos valores possuem prefixos comuns entre si. Com esta técnica os prefixos comuns de termos textuais sequenciais não necessitam serem replicados. Como a Tabela de Mapeamento de Códigos Rótulo-Valor é um índice textual, esta técnica é empregada no armazenamento desta tabela com o objetivo de reduzir o tamanho resultante.

Por exemplo, supondo existir o conjunto de termos indexados de acordo com o exemplo mostrado na Figura 45a, então as informações armazenadas após compressão são listadas na Figura 45b. Para cada termo, ao invés de ser armazenado o prefixo coincidente

com o termo anterior é armazenado um valor  $i$  que indica que o prefixo do termo vai até o  $i$ -ésimo caractere do termo anterior. Conforme ilustrado no exemplo, ao invés de ser armazenada a sequência textual  $[@nome="Shannon"]$  é armazenada a sequência  $nnon"]$  mais um valor indicador  $11$  que informa que o termo tem como prefixo os onze primeiros caracteres do termo anterior  $[@nome="Shakespeare"]$ .

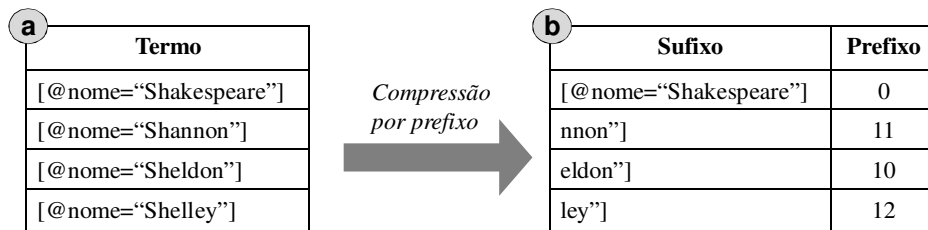


**Figura 45: Aplicação de compressão por prefixo**

ii) A técnica de *compressão por intervalo* [Manning08] é empregada no armazenamento de arranjo de valores inteiros monotônicos crescentes (que preservam a relação de ordem crescente) motivado pela ideia chave de que o arranjo de valores de diferença entre valores sequenciais são menores e relativamente pequenos em relação aos próprios valores originais. Já que os códigos inteiros e ponteiros de um mesmo bloco, bem como, as listas de identificadores de documentos são arranjos de valores inteiros monotônicos crescentes, então a técnica de compressão por intervalo é empregada no armazenamento dos mesmos.

Por exemplo, tomando o resultado final da ordenação dos caminhos mostrados anteriormente na Tabela 11, ao invés de serem armazenados os códigos inteiros listados na Figura 46a são armazenados os valores da diferença entre códigos inteiros, conforme mostrados na Figura 46b. Perceba que os valores armazenados são geralmente menores do que os códigos inteiros originais. Logo, gastam menos espaço de armazenamento.

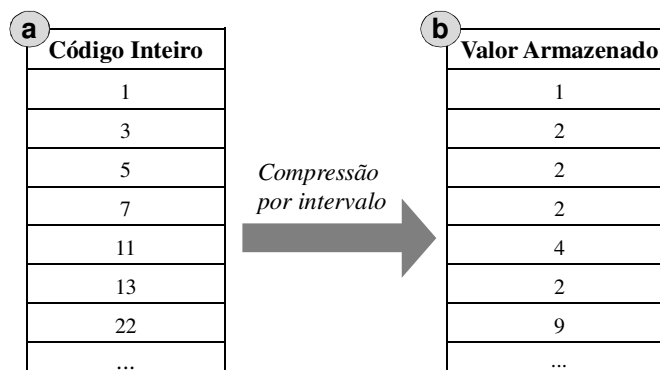


**Figura 46: Aplicação de compressão por intervalo**

iii) Ocorre desperdício de armazenamento quando se emprega a codificação de comprimento fixo (ex: codificação em 4 bytes para inteiros) para armazenar números pequenos. Isto decorre da alocação de mais bytes do que o necessário para armazenar tais dados. Diminuir o comprimento fixo também não é uma solução adequada, pois inviabiliza a representação de números que exigem mais bytes do que o comprimento utilizado.

Neste contexto, de modo a reduzir o espaço de armazenamento, bem como permitir o armazenamento de números grandes, utiliza-se a *codificação em bytes de tamanho variável* [Manning08]. Esta codificação é eficiente em termos de tempo de execução. Através desta codificação, números pequenos (no intervalo de 0 até 127) ocupam somente um byte. Como os arranjos comprimidos com a técnica de compressão por intervalo geram números pequenos, estes também são codificados em bytes de tamanho variável.

A Figura 47a mostra os endereços para o arquivo de resultados de pesquisa gerado para o exemplo apresentado anteriormente na Tabela 5. No índice final, como os arquivos são escritos sequencialmente, os endereços de resultado de pesquisa aumentam de valor a partir da primeira chave de índice até a última chave de índice, formando assim um conjunto monotônico crescente em que pode ser aplicada a compressão por intervalo. Assim, na Figura 47a temos que todo endereço  $e_i$  é numericamente menor do que todo endereço  $e_j$  para  $i < j$ .

| a | Caminho Indexado                                    | Ponteiro | b | Valor Armazenado |
|---|-----------------------------------------------------|----------|---|------------------|
|   | /editora                                            | $e_1$    |   | $e_1$            |
|   | /editora[@nome="SparkNotes"]                        | $e_2$    |   | $e_2 - e_1$      |
|   | /editora[@cidade="Oxford"]                          | $e_2$    |   | 0                |
|   | /editora/livro                                      | $e_3$    |   | $e_3 - e_2$      |
|   | /editora[@nome="SparkNotes"]/livro                  | $e_3$    |   | 0                |
|   | /editora[@cidade="Oxford"]/livro                    | $e_3$    |   | 0                |
|   | /editora/livro[@nome="Hamlet"]                      | $e_4$    |   | $e_4 - e_3$      |
|   | /editora/livro[@nome="Othello"]                     | $e_5$    |   | $e_5 - e_4$      |
|   | /editora[@nome="SparkNotes"]/livro[@nome="Hamlet"]  | $e_6$    |   | $e_6 - e_5$      |
|   | /editora[@cidade="Oxford"]/livro[@nome="Hamlet"]    | $e_6$    |   | 0                |
|   | /editora[@nome="SparkNotes"]/livro[@nome="Othello"] | $e_7$    |   | $e_7 - e_6$      |
|   | /editora[@cidade="Oxford"]/livro[@nome="Othello"]   | $e_7$    |   | 0                |

**Figura 47: Valores de endereço armazenados**

Neste cenário, se duas chaves de índice sequenciais são associadas a ponteiros que endereçam para o mesmo resultado de pesquisa, isto é, reusam a mesma lista de resultados de

pesquisa, então, através da compressão por intervalo, a diferença entre o segundo ponteiro e primeiro tem o valor 0 (zero). Ao armazenar esse segundo ponteiro com a compressão por codificação em bytes de tamanho variável o mesmo ocupará somente um byte, o qual é o tamanho ótimo de armazenamento. Para o exemplo da Figura 47a, os valores de endereço armazenados são mostrados na Figura 47b, percebe-se que os endereços com valor 0 (zero) correspondem aos resultados de pesquisa reusados (destacados com sobreposição em cinza).

iv) A técnica BranchGuide é projetada para recuperar entidades descritas por documentos e não por partes de um determinado documento, desta forma, idealmente, espera-se que uma fonte de dados indexada seja composta de uma grande coleção formada por pequenos ou médios documentos XML e não, ao contrário, por uma pequena coleção de documentos formada por documentos XML extensos. Desta forma, os códigos de eixo de atributo irmão atribuídos individualmente para cada documento não tendem a ter um crescimento ao infinito, mas ao contrário, um crescimento limitado.

Neste cenário torna-se ideal o emprego de um esquema de codificação de bytes de tamanho variável para unidades menores do que o byte. Neste caso, a compressão em bytes de tamanho variável pode ser remodelada e também pode ser aplicada em maiores ou menores unidades de bytes. Por exemplo, uma unidade menor do que o byte é o *nibble* [Manning08]. Um nibble é formado por uma palavra de 4 (quatro) bits.

A atribuição de códigos de eixo de atributo irmão é crescente e monótona de forma que se pode aplicar a compressão por intervalo sob estas informações. Como os códigos de eixo irmão atribuídos por documento são geralmente pequenos espera-se que a diferença entre os mesmos sejam valores pequenos próximos de zero. Neste caso, a *codificação em nibbles de tamanho variável* permite que números bastante pequenos (no intervalo de 0 até 7) ocupem somente um nibble (4 bits). Sendo assim, em arranjos de códigos de atributo de eixo irmão primeiramente é aplicada a compressão por intervalo e, por fim, são armazenados utilizando codificação em nibbles de tamanho variável.

### 5.3 Considerações Finais

É importante ressaltar que a função de *hash* perfeito projetada para a técnica BranchGuide (detalhada na subseção 5.1.1) tem uma importância central na concretização do projeto do algoritmo de indexação proposto pois: i) A função de *hash* perfeita proposta possibilita mapear sem colisões uma expressão de caminho em forma textual para um identificador numérico exclusivo. Se existissem colisões no mapeamento, isto é, se a função de *hash* não

fosse perfeita, seria possível que dois caminhos diferentes pudessem ser mapeados para códigos inteiros iguais o que iria causar erros de precisão de consulta ao índice. *ii)* Um segundo fator é que a ordenação dos códigos mapeados para as expressões de caminho possibilita que os caminhos sejam ordenados conforme especificação do projeto do algoritmo de indexação proposto, esta em que, caminhos associados à vértices (do grafo de índice) com probabilidade de serem replicados são ordenados para serem armazenados sequencialmente.

Dado a importância da função de *hash* perfeito proposta é relevante considerar de onde veio a ideia inicial para a motivação, definição e modelagem da mesma. A função de *hash* perfeito proposta mapeia uma sequência de números decimais para um exclusivo número decimal. A motivação para a definição da função da *hash* perfeito proposta é baseada nos métodos de transformação de base (uma base qualquer) para a base decimal [Sinha11]. Por exemplo, no caso da transformação de números na base 2 (especificamente para este exemplo, número binários com no máximo três dígitos) para a base decimal temos, de acordo com as expressões definidas na subseção 5.1.1, que:

- i)*  $T = 3$  é o número máximo de dígitos de um número binário neste exemplo.
- ii)* O conjunto de quantidade de distintos números por ordem  $= Q = \{2, 2, 2\}$ , já que cada dígito pode assumir o valor zero (0) ou um (1).
- iii)* Então, o conjunto de base  $= B = \{2^*2, 2, 1\} = \{4, 2, 1\} = \{2^2, 2^1, 2^0\}$ .

Por exemplo, temos que o processo de transformação do número binário 101 (base 2) para a base decimal (base 10) é dado por [Sinha11]:

$$101_2 = 2^2 * 1 + 2^1 * 0 + 2^0 * 1 = 4 + 0 + 1 = 5_{10}.$$

Além disso, de acordo com a definição da função de *hash* perfeito (mostrada na Figura 40) tem-se que:

$$h(P=(1,0,1)) = \sum_{j=(1 \dots T)} P_j * B_j = 1 * 4 + 0 * 2 + 1 * 1 = 4 + 0 + 1 = 5_{10}.$$

Ou seja,  $h(P=(1,0,1)) = 5_{10} = 101_2$ . Isto é, tomando o conjunto de sequências numéricas em que cada sequência tem três dígitos associados, e, além disso, cada dígito pode assumir o valor zero (0) ou o valor um (1) temos que, neste cenário, aplicar a função de *hash* perfeito sob uma sequência numérica equivale a obter o resultado da transformação da sequência binária equivalente para um exclusivo número decimal.

Os métodos de transformação de base constituem a principal motivação para a definição da função de *hash* proposta. Apesar disso, a principal diferença entre os métodos de

transformação de base para o cálculo de mapeamento empregado na função de *hash* proposta é que no caso dos métodos de transformação de uma base qualquer para a base decimal temos que a variação de valores para cada item de uma sequência numérica sempre é fixo. Isto é, no caso da transformação de base 2 para a base decimal, em cada dígito de um número binário pode ocorrer a variação entre dois valores  $\{0,1\}$ . Já no caso das expressões de caminho indexadas (isto é, sequências numéricas indexadas) pela técnica BranchGuide, o conjunto de valores que pode ser atribuído para cada item das sequências numéricas indexadas pode variar, como no exemplo ilustrado na Tabela 8.

A modelagem da função de *hash* perfeito proposta trata esta possibilidade de variação através da definição do cálculo de um conjunto de base específico que depende das informações (a quantidade de diferentes nomes indexados em cada nível) da amostra. Isto possibilita que um conjunto de sequências numéricas seja perfeitamente mapeado (sem colisões) para a base decimal mantendo uma correspondência quantitativa, isto é, para cada sequência existe um valor quantitativo (decimal) exclusivo correspondente.

## 6 Avaliação Experimental

Com o objetivo de avaliar a técnica BranchGuide foram realizados diversos experimentos. O objetivo geral dos experimentos é medir a eficiência da técnica BranchGuide e compará-la com uma técnica de indexação baseada em interseção estrutural.

Primeiramente, é importante ressaltar que técnicas de indexação com perda de precisão podem retornar resultados insuficientes para comparação com técnicas de indexação sem perdas, em outras palavras, não é adequado contrastar estes resultados porque os resultados de pesquisa imprecisos podem conter documentos que não satisfazem a consulta especificada, ao contrário, técnicas sem perda de precisão só incluem nos resultados os documentos que realmente satisfazem a consulta.

Portanto, para os experimentos apresentados neste capítulo, uma técnica de indexação sem perdas de precisão é escolhida para comparação. Como anteriormente discutido no Capítulo 2, os índices de sequência apresentam perda de precisão, por isso, tais técnicas de indexação (como por exemplo, ViST [Wang03]) não são adequadas para comparação experimental. Em contrapartida, as técnicas de interseção estrutural não apresentam perda de precisão, logo, uma técnica de interseção estrutural é utilizada como base de comparação nos experimentos realizados.

Dentre as propostas baseadas em interseção estrutural, destaca-se o eXist [Meier02]. O eXist é um sistema *open source* de banco de dados XML que emprega técnicas de interseção estrutural similares as que foram inicialmente proposta no sistema XISS [Li01], o qual é um trabalho precursor da aplicação de técnicas de interseção estrutural para descoberta de documentos XML. O eXist pode ser visto como uma evolução do XISS, já que o eXist adota um esquema mais eficiente de identificação de elementos, denominado *DLN (Dynamic Level Numbering)* [Timo04]. Outra característica importante é que o eXist emprega uma técnica nativa de indexação, isto é, não adota outro modelo de banco de dados como modelo básico de armazenamento dos índices, tais como banco de dados relacional ou orientado a objetos.

Sendo assim, um protótipo de implementação da técnica BranchGuide foi empregado para gerar resultados a serem contrastados com os resultados gerados pelo eXist. Estes resultados são apresentados e discutidos nas próximas seções.

## 6.1 Amostras para Avaliação

Duas coleções com diferentes características foram indexadas através do eXist bem como utilizando a técnica BranchGuide. As propriedades destas coleções são descritas nas subseções a seguir.

### 6.1.1 DBLP

A amostra DBLP (*Digital Bibliography & Library Project*) [Ley09] é amplamente utilizada como amostra de avaliação para sistemas de banco de dados para XML e técnicas de indexação para dados semi-estruturados. A amostra DBLP está disponível publicamente como um único e amplo documento XML. A fim de simular vários documentos, o único arquivo da DBLP foi dividido em vários arquivos em que cada um descreve uma entidade (ex: artigo publicado, livro ou tese). Tais documentos são relativamente curtos em relação ao tamanho e tem em média dois atributos XML por documento, desta forma, na indexação destes documentos são extraídas poucas expressões com ramificação. Uma visão mais detalhada de uma amostra com 200.000 documentos da DBLP é mostrada na Tabela 12.

**Tabela 12: Propriedades da amostra DBLP**

| DBLP                                                               |        |
|--------------------------------------------------------------------|--------|
| Quantidade Média de Atributos por Documento                        | 2,072  |
| Quantidade Média de Elementos/Atributos XML por Documento          | 22,579 |
| Quantidade Média de Linhas por Documento                           | 15,248 |
| Quantidade Média de Caminhos com Ramificação Gerados por Documento | 57,736 |

### 6.1.2 Amostra X-ARM

Em um processo de *Desenvolvimento Baseado em Componentes*, diversos tipos de artefatos podem ser gerados, e, consequentemente, armazenados em um repositório. Apesar disso, no repositório, geralmente, os artefatos armazenados não são autodescritivos, e, além disso, podem ter diferentes formatos: desde documentações textuais e diagramas até código fonte. Logo, a fim de tratar esta dificuldade, os modelos de representação para descrição de componentes provêm um método para a descrição de metadados de artefatos de software;

desta forma, estes metadados podem ser indexados por sistemas de recuperação de informação, para assim, potencializar o reuso de artefatos.

Nesta avaliação experimental, empregamos o modelo de representação de artefatos de software X-ARM [Schuenck06] para gerar uma amostra de documentos de descrição de artefatos de software. O X-ARM adota um modelo de descrição baseado em XML.

É importante ressaltar que o X-ARM permite descrever não apenas produtos finais, ou seja, implementações de componentes, mas também diversos tipos de artefatos produzidos durante o processo de desenvolvimento; dentre tais artefatos, em especial, destacamos as informações sobre especificações, implementações de componentes, interfaces e operações. Este grupo de tipos de artefatos foi utilizado na geração randômica da amostra. Cada artefato descrito com o X-ARM tem um identificador único e é armazenado através de um documento XML. O reuso de artefatos é possível, pois uma descrição de artefato pode referenciar outros artefatos do repositório através dos identificadores.

Na geração de amostras de descrições X-ARM, vários artefatos dos tipos *Interface Independente*, *Interface Dependente*, *Especificação Independente*, *Especificação Dependente* e *Implementação de Componente* são criados com identificadores únicos. As possíveis relações de reuso existentes entre estes artefatos são mostrados na Figura 48.

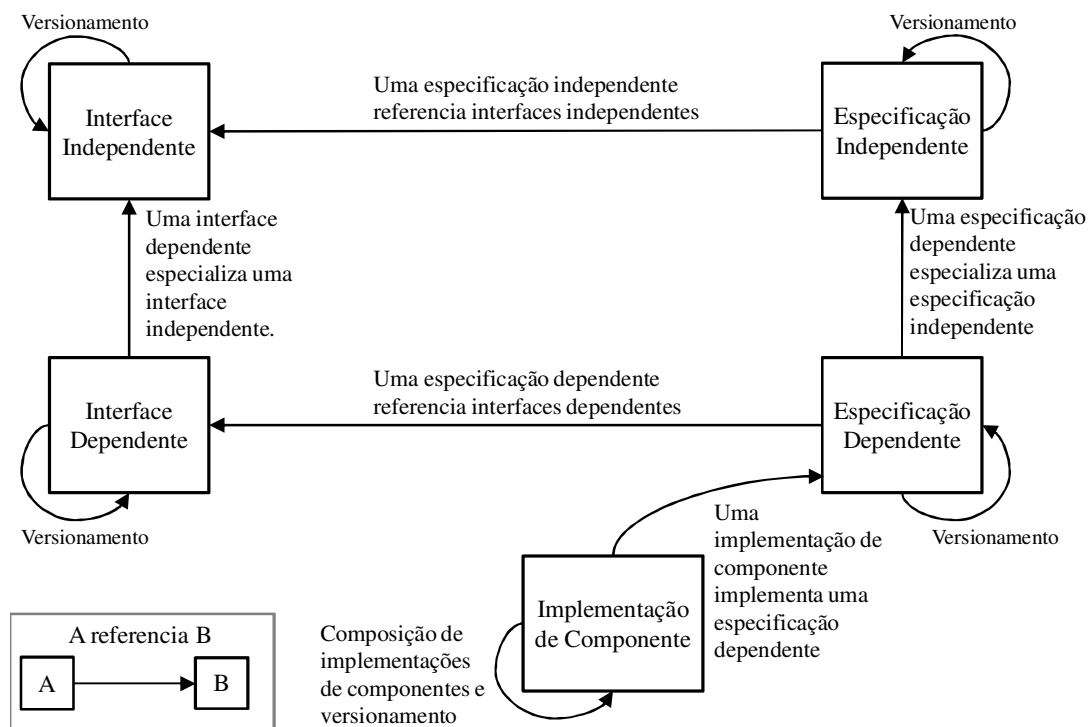


**Figura 48: Relações de reuso geradas na amostra X-ARM**

Por exemplo, uma especificação de componentes pode referenciar um conjunto de interfaces. A indicação de referência de um dado artefato A para um dado artefato B é descrita no documento XML através de valores de atributos XML específicos definidos pelo modelo de representação X-ARM. Além disso, para os artefatos de *Interface*, nomes de operações e parâmetros também são descritos através de valores textuais de atributos XML.

Apesar da amostra ser gerada artificialmente de forma randômica é necessário que alguns critérios sejam definidos para a atribuição de valores textuais de atributos a fim de não gerar enviesamento dos resultados do experimento. Para isto, empregam-se critérios para a distribuição de valores textuais dos atributos da amostra. Como apontado por Manning [Manning08], a distribuição de termos textuais em uma coleção segue as seguintes leis:

i) *Lei de Heap: novos termos surgem conforme mais documentos são adicionados na coleção.* Por exemplo, uma amostra em que um conjunto limitado de termos é distribuído conforme novos documentos são criados **não** satisfaz esta lei.

ii) *Lei de Zipf: O valor de frequência de ocorrência entre o mais frequente termo da coleção até o menos frequente termo da coleção decresce rapidamente.* Por exemplo, em amostras em que todos os documentos são diferentes, temos a mesma frequência de ocorrência para todo termo da coleção, logo, **não** satisfazendo esta lei.

Neste sentido, baseado nas leis anteriormente citadas, ao atribuir aleatoriamente valores textuais a atributos XML, os seguintes critérios de distribuição são definidos. Tomando as coleções  $C_1, C_2, \dots, C_n$ , e dado que a quantidade de documentos da amostra  $C_j$  é maior do que a quantidade de documentos da amostra  $C_i$  para todo  $j > i$ . Com isto, se para a coleção  $C_i$  um conjunto  $N_i$  de termos textuais é uniformemente distribuído, então, para a coleção  $C_{i+1}$  um conjunto  $N_{i+1}$  de termos textuais deve ser uniformemente distribuído tal que as seguintes condições sejam verdadeiras:

i) *Sendo  $|N_i|$  o número de termos distintos distribuídos para a coleção  $C_i$  e  $|N_{i+1}|$  o número de termos distintos distribuídos para a coleção  $C_{i+1}$ , então  $|N_{i+1}| = |N_i| + \text{constante}$ .*

ii) *O conjunto de termos  $N_i$  está contido em  $N_{i+1}$ .*

Desta forma, como uma nova quantidade (sempre adicionada de uma constante) de termos distintos pode ser distribuída para uma coleção com mais documentos, então, novos termos surgem conforme mais documentos são adicionados, assim, verificando a *Lei de Heap*. Além disso, conforme o número de termos distribuídos cresce para coleções com mais documentos, então, os termos que foram distribuídos anteriormente em amostras menores e

que ocorrem novamente em amostras maiores têm uma probabilidade de ocorrência maior; já, ao contrário, termos que somente surgiram em amostras menores têm uma probabilidade de ocorrência menor. Isto gera um valor de frequência não igual e decrescente para o conjunto global de termos, assim, verificando a *Lei de Zipf*.

A amostra X-ARM é concebida para expor a técnica BranchGuide para um cenário de pior caso, em que os documentos são mais longos em média e geram uma quantidade maior de caminhos com ramificação indexados (a cerca de quinze vezes mais do que os documentos da amostra DBLP), pois os documentos da amostra X-ARM têm em média uma grande quantidade de atributos. Assim, no processo de indexação, enquanto que a amostra X-ARM gera uma grande quantidade de caminhos longos com ramificação, a amostra DBLP deriva somente uma pequena quantidade de caminhos com ramificação. Uma visão mais detalhada de uma amostra com 200.000 documentos da amostra X-ARM é apresentada na Tabela 13.

**Tabela 13: Propriedades da amostra X-ARM**

| X-ARM                                                              |          |
|--------------------------------------------------------------------|----------|
| Quantidade Média de Atributos por Documento                        | 56,5511  |
| Quantidade Média de Elementos/Atributos XML por Documento          | 84,2191  |
| Quantidade Média de Linhas por Documento                           | 46,4667  |
| Quantidade Média de Caminhos com Ramificação Gerados por Documento | 804,1199 |

## 6.2 Configuração do Experimento

Nesta seção são detalhadas propriedades do experimento as quais incluem: fatores e níveis do experimento, métricas avaliadas no experimento, ambiente de execução do experimento, e por fim, o tipo de análise estatística empregada no experimento.

i) Fatores e níveis do experimento são listados na Tabela 14.

**Tabela 14: Fatores e níveis**

| Fatores                  | Níveis                                                                                |
|--------------------------|---------------------------------------------------------------------------------------|
| Amostras                 | {DBLP, Amostra X-ARM}                                                                 |
| Quantidade de Documentos | {10000, 20000, 30000, 40000, 50000, 100000, 150000, 200000}                           |
| Técnicas de Indexação    | {Técnica BranchGuide, Técnica de interseção estrutural implementada no sistema eXist} |
| Consultas                | {Tabela 15 (para DBLP), Tabela 18 (para amostra X-ARM)}                               |

ii) Métricas avaliadas: Tempo de Execução de Consultas, Tamanho de Índice e Tempo de Indexação.

iii) Ambiente de execução: O protótipo de implementação da técnica BranchGuide bem como o eXist executam sob o ambiente Java. Os dois sistemas avaliados foram executados em uma mesma máquina *Intel Core 2 Quad Q6600 2.4GHz, 2GB RAM*.

iv) Análise estatística: Para cada consulta avaliada, executaram-se medições até ser obtida uma média com erro percentual de 2,5%. Neste caso, para estimativa de erro adotou-se a distribuição de *T-Student* [Jain91] com nível de confiança de 95%.

## 6.3 Experimentos com a Amostra DBLP

### 6.3.1 Tamanho de Índice

Como pode ser observado na Figura 49, os tamanhos de índice gerados pela técnica BranchGuide são drasticamente menores do que os produzidos pelo eXist. Mais importante, como uma evidência da escalabilidade da técnica BranchGuide, quanto maior é a quantidade de documentos, maior é o ganho em termos do tamanho de índice em relação ao eXist. Além disso, vale notar que o índice gerado pela técnica BranchGuide tem tamanho muito próximo ao da amostra original. Isto é um resultado esperado, pois como a amostra DBLP tem poucas ramificações indexadas isto gera uma complexidade de armazenamento aproximadamente linear para o índice gerado.

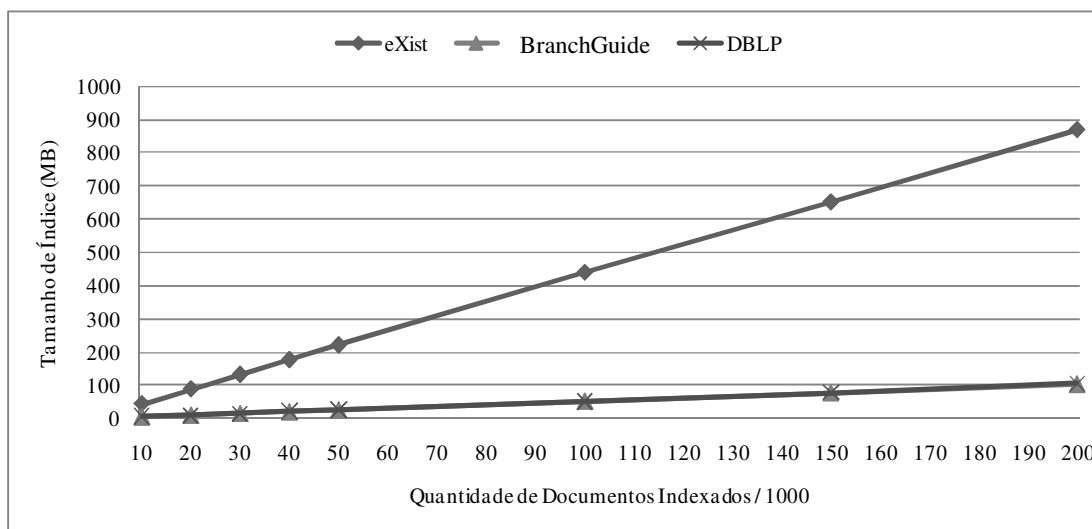


Figura 49: Tamanho de índice: DBLP

### 6.3.2 Tempo de Execução de Consultas

Já que a amostra DBLP não gera muitas ramificações, para medição é avaliada uma consulta com uma ramificação e uma consulta simples. As consultas estão listadas na Tabela 15.

**Tabela 15: Consultas para a amostra DBLP**

| Identificador | Consulta                                                               | Comentário                        |
|---------------|------------------------------------------------------------------------|-----------------------------------|
| $Q_1$         | /dblp                                                                  | Consulta simples                  |
| $Q_2$         | /dblp/inproceedings[ @mdate="2005-07-07"]/author[#text="Dong-Hong Ji"] | Consulta com uma (1) ramificação. |

Como pode ser observado no gráfico apresentados na Figura 50, os tempo de execução de consulta para  $Q_1$  ao empregar a técnica BranchGuide são consideravelmente menores do que os tempos de execução no eXist. Isto se dá pelo fato que, na técnica BranchGuide, a execução de uma consulta simples implica em um simples acesso ao índice para leitura dos identificadores de documentos e códigos de eixo de atributo irmão. Já no eXist, a execução da consulta  $Q_1$  retorna um conjunto de identificadores de elementos codificados no sistema de classificação Dewey [Timo04]. Estes experimentos mostram que o esquema de identificação empregado no eXist gera uma ligeira sobrecarga no acesso das listas de resultados de pesquisa.

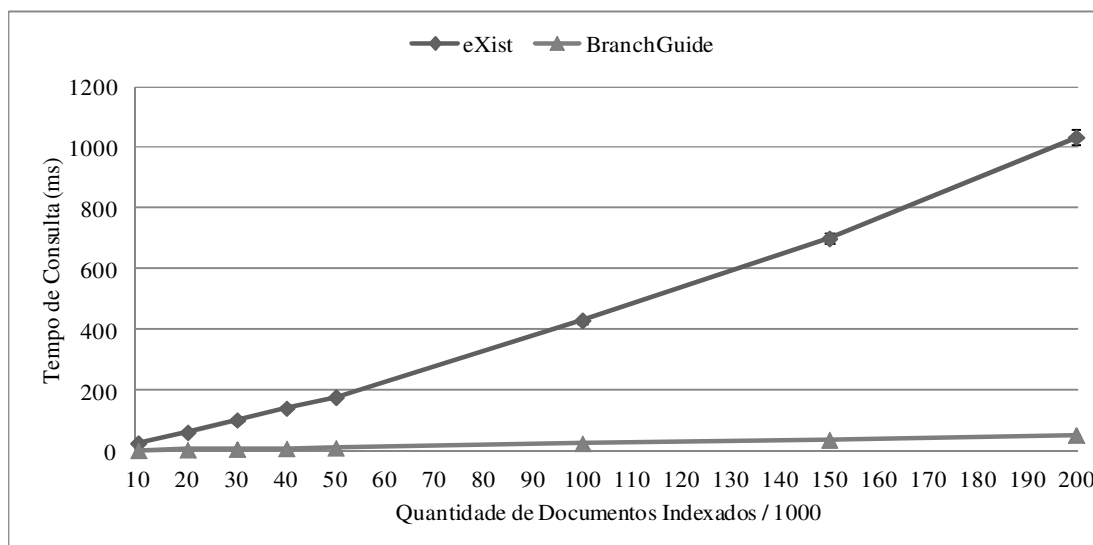


**Figura 50: Comparação de tempo de execução da consulta  $Q_1$**

Perceba também que na Figura 50, os tempos de consulta de  $Q_1$  variam com o aumento do número de documentos. Isto ocorre porque o tamanho da lista de resultados (que contém a quantidade de ocorrências retornadas) da consulta  $Q_1$  varia com o número de documentos indexados, já que a consulta  $Q_1$  retorna os identificadores de todos os

documentos indexados. Como o tempo de recuperação dos identificadores de documentos retornados é somado ao tempo total de execução da consulta, então o tempo de recuperação total aumenta conforme o número total de documentos indexados aumenta.

Já em relação à consulta  $Q_2$ , como pode ser observado no gráfico apresentado na Figura 51, os tempos de execução da consulta  $Q_2$  ao empregar a técnica BranchGuide são consideravelmente menores do que os tempos de execução de consulta do eXist. Isto ocorre uma vez que na técnica BranchGuide a execução de uma consulta com ramificação implica em um simples acesso ao índice. Por outro lado, no eXist, uma consulta com ramificação executa várias etapas de interseção estrutural para obter o resultado final da consulta. A interseção estrutural gera uma sobrecarga acentuada na execução da consulta, implicando assim, em uma relevante diferença entre os tempos de consulta gerados pelas técnicas.

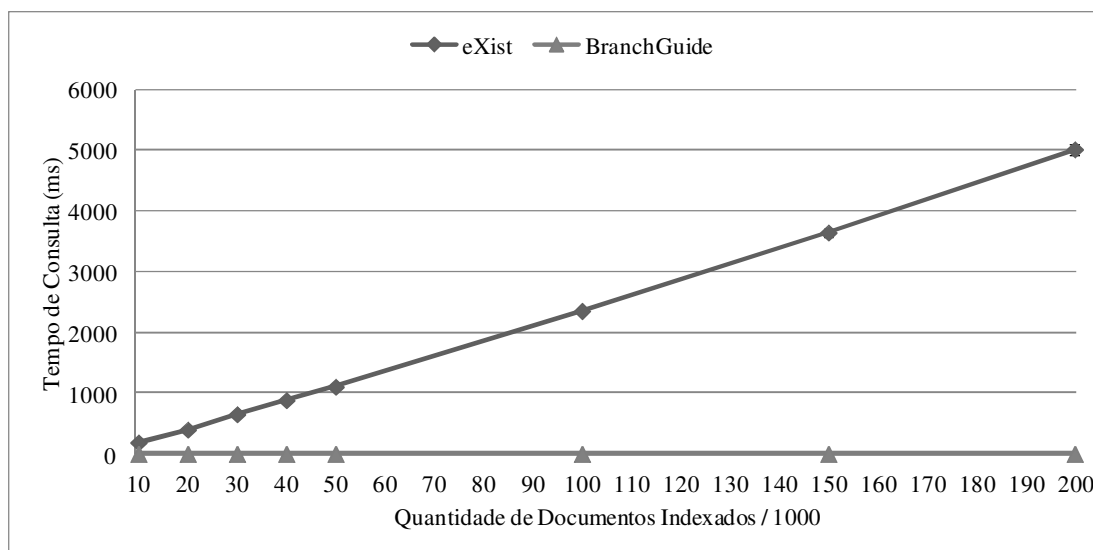


**Figura 51: Comparação de tempo de execução da consulta  $Q_2$**

Por fim, as Tabelas 16 e 17 detalham os valores de tempo de execução das consultas  $Q_1$  e  $Q_2$ , respectivamente.

**Tabela 16: Tempos de consulta para  $Q_1$**

| Quantidade de Documentos | eXist (ms) | Erro (ms): eXist | BranchGuide (ms) | Erro (ms): BranchGuide |
|--------------------------|------------|------------------|------------------|------------------------|
| 10000                    | 26,016     | 0,323            | 2,094            | 0,052                  |
| 20000                    | 61,554     | 0,604            | 4,347            | 0,108                  |
| 30000                    | 102,516    | 1,011            | 6,343            | 0,158                  |
| 40000                    | 141,473    | 1,134            | 8,590            | 0,214                  |
| 50000                    | 177,802    | 1,587            | 10,610           | 0,264                  |
| 100000                   | 431,160    | 10,751           | 25,206           | 0,630                  |
| 150000                   | 700,889    | 17,440           | 37,604           | 0,939                  |
| 200000                   | 1034,525   | 25,791           | 52,583           | 1,3145                 |

Tabela 17: Tempos de consulta para  $Q_2$ 

| Quantidade de Documentos | eXist (ms) | Erro (ms): eXist | BranchGuide (ms) | Erro (ms): BranchGuide |
|--------------------------|------------|------------------|------------------|------------------------|
| 10000                    | 190,762    | 1,074            | 0,284            | 0,001                  |
| 20000                    | 402,760    | 9,869            | 0,313            | 0,005                  |
| 30000                    | 656,287    | 11,015           | 0,286            | 0,005                  |
| 40000                    | 887,787    | 19,617           | 0,244            | 0,001                  |
| 50000                    | 1107,179   | 21,862           | 0,346            | 0,002                  |
| 100000                   | 2354,725   | 42,817           | 0,348            | 0,001                  |
| 150000                   | 3645,893   | 66,084           | 0,302            | 0,001                  |
| 200000                   | 5013,008   | 97,287           | 0,257            | 0,001                  |

### 6.3.3 Tempo de Indexação

O gráfico mostrado na Figura 52 apresenta o tempo de indexação da amostra DBLP utilizando a técnica BranchGuide e o eXist. Os tempos de indexação da técnica BranchGuide variam no máximo em aproximadamente *13 minutos* e no mínimo por aproximadamente *26 segundos* em relação com os tempos de indexação obtidos no eXist. Esta diferença é esperada para a técnica BranchGuide pois enquanto que o eXist extrai nomes de elementos e atributos dos documentos, a técnica BranchGuide extrai caminhos com ramificação. A quantidade de caminhos com ramificação extraídos tem uma complexidade maior do que linear [Goldman97], enquanto que o número de nomes extraídos é linearmente dependente à quantidade de nomes existentes na amostra. Ou seja, o número de caminhos com ramificação extraídos sempre vai ser maior ou igual do que a quantidade de nomes existentes.

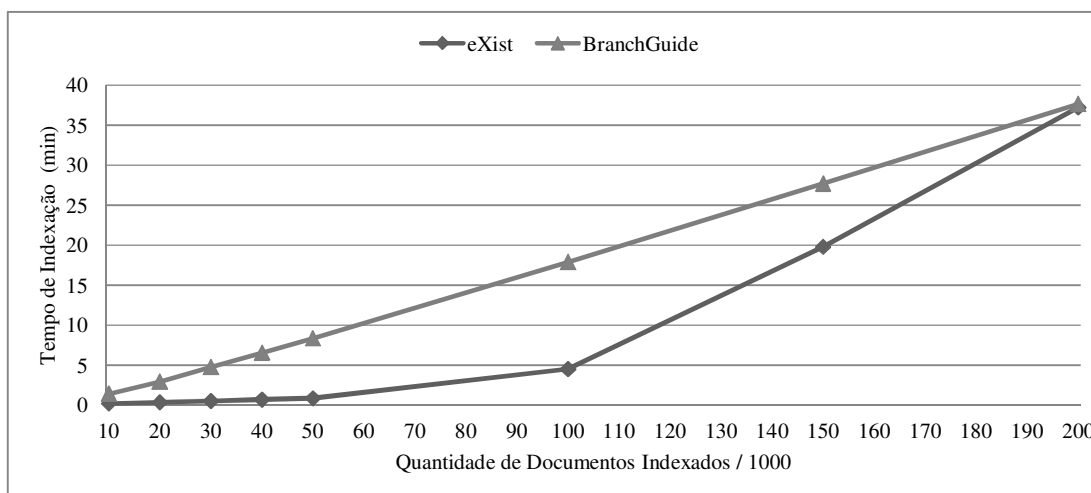


Figura 52: Tempo de indexação: DBLP

Neste caso, como a amostra DBLP tem uma baixa quantidade de ocorrências de caminhos com ramificação, então a diferença de tempo necessário para a indexação com a técnica BranchGuide torna-se pequena em relação com o tempo de indexação pelo eXist.

## 6.4 Experimentos com a Amostra X-ARM

### 6.4.1 Tamanho de Índice

Como pode ser visto na Figura 53, os tamanhos de índice gerados pela técnica BranchGuide são menores do que os produzidos pelo eXist. Além disso, como uma evidência da escalabilidade da técnica BranchGuide, quanto maior é a quantidade de documentos, maior é o ganho em termos do tamanho de índice em relação ao eXist.

Esses resultados em relação aos tamanhos de índice são obtidos principalmente como uma consequência do emprego do reuso das listas de resultado de pesquisa, evitando assim que informações idênticas associadas a vértices duplicados não sejam armazenadas. Temos que a amostra X-ARM gera uma grande quantidade de caminhos com ramificação indexados, desta forma, o reuso das listas de resultados de pesquisa reduz o custo de armazenamento das informações armazenadas relacionadas aos caminhos com ramificação, evitando assim, um crescimento elevado do tamanho de índice.

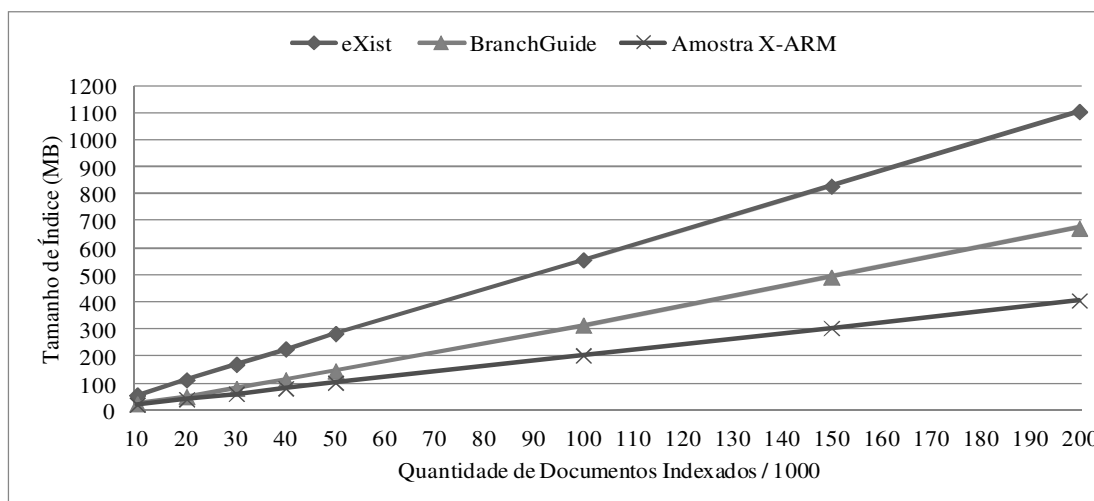


**Figura 53: Tamanho de índice: X-ARM**

### 6.4.2 Tempo de Execução de Consultas

Já que a extração de caminhos a partir da amostra X-ARM gera muitas ramificações, para medição, além de consultas simples também é avaliado um conjunto de consultas com ramificação tal que é variado o número de ramificações. Além disso, o desempenho do mecanismo de recuperação de informação por similaridade é avaliado através da variação da quantidade de operadores de conjunção em expressões de consulta. As consultas avaliadas estão listadas na Tabela 18.

**Tabela 18: Consultas para a amostra X-ARM**

| Identificador | Consulta                                                                                                                                                                | Comentário                                       |
|---------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------|
| $Q_3$         | /asset                                                                                                                                                                  | Consulta simples                                 |
| $Q_4$         | /asset[@owner="Trusecure-770"]                                                                                                                                          | Consulta simples                                 |
| $Q_5$         | /asset[@owner="BEA-1008"]/interface/operation[@name="notifyCustomer-317"]                                                                                               | Consulta com uma (1) ramificação.                |
| $Q_6$         | /asset[@owner="PMI-1285"]/interface/operation[@name="getEmailId-92"]/parameter[@name="date33"]                                                                          | Consulta com duas (2) ramificações.              |
| $Q_7$         | /asset[@owner="File Maker-321" and @date="1966-05-10"]/interface/operation/parameter[@name="price27" and @type="Boolean"]                                               | Consulta com dois (2) operadores de conjunção.   |
| $Q_8$         | /asset[@owner="Network General-1142" and @date="1961-08-09" and @language="hi-57"]/interface/operation/parameter[@name="value74" and @type="void" and @direction="out"] | Consulta com quatro (4) operadores de conjunção. |

Como pode ser observado nos gráficos mostrados nas Figuras 54, 55, 56 e 57, os tempos de consulta para  $Q_3$ ,  $Q_4$ ,  $Q_5$  e  $Q_6$  utilizando a técnica BranchGuide são consideravelmente menores do que os tempos de consulta gerados utilizando o eXist. Isto se dá pelo fato que, com a técnica BranchGuide, a execução das consultas simples  $Q_3$  e  $Q_4$  são efetuadas através de um simples acesso ao índice. Além disso, apesar da quantidade de ramificações de  $Q_5$  e  $Q_6$  serem diferentes, o tempo de execução destas consultas mantém-se a cerca de um valor aproximado, pois, independentemente do número de ramificações, é executado um único acesso ao índice e o resultado é recuperado diretamente sem processamento adicional. Por outro lado, no eXist, quanto mais ramificações existirem na consulta, mais etapas de interseção estrutural são executadas para obter o resultado final da consulta, e assim, degrada-se o desempenho resultante.

Vale ressaltar que, utilizando a técnica BranchGuide, os tempos de consulta de  $Q_3$  variam com o aumento do número de documentos, enquanto que os tempos de consulta de  $Q_4$ ,  $Q_5$  e  $Q_6$  permanecem a cerca de um mesmo valor aproximado. Isto ocorre porque o tamanho da lista de resultados de pesquisa (isto é, a quantidade de identificadores de documentos retornados) da consulta  $Q_3$  varia bastante com o número de documentos indexados.

A consulta  $Q_3$  tem baixa restrição de seleção, retornando sempre os identificadores de todos os documentos da fonte de dados. Logo, o tempo de recuperação da lista de resultados de pesquisa contribui ligeiramente com o tempo total de execução da consulta. Ao contrário, as lista de resultados de  $Q_4$ ,  $Q_5$  e  $Q_6$  mantêm tamanhos aproximados conforme o número total de documentos indexados aumenta.

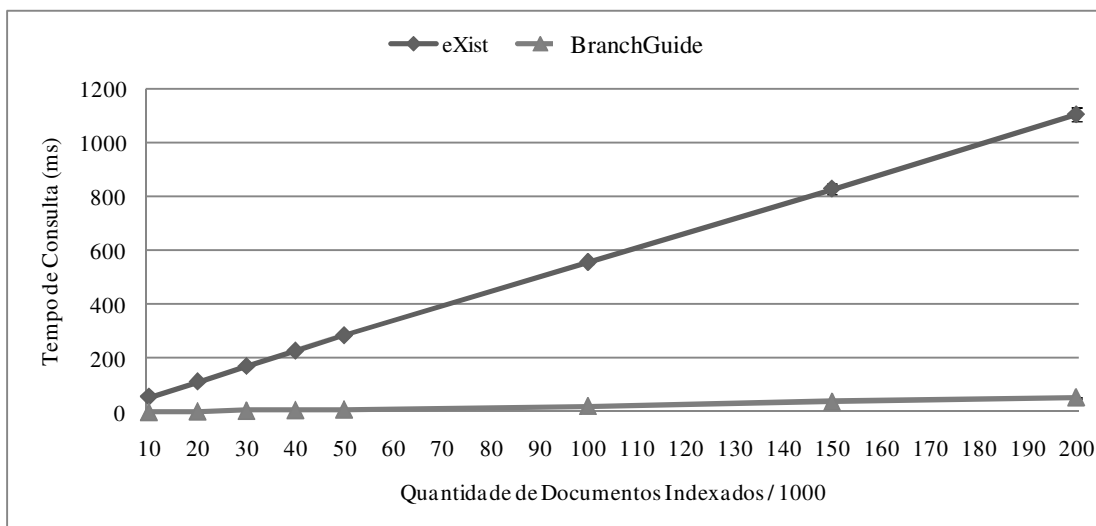


Figura 54: Comparação de tempo de execução da consulta  $Q_3$

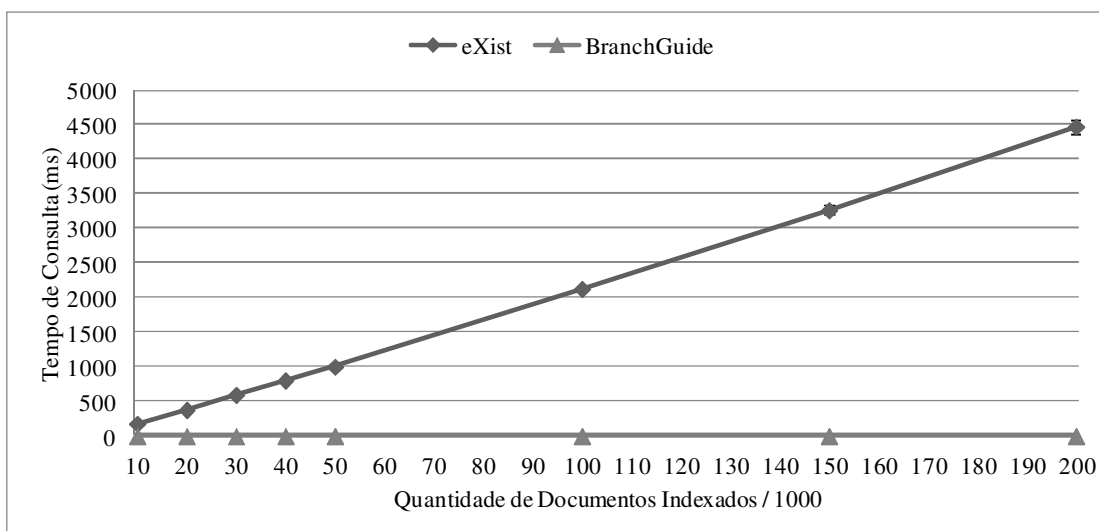


Figura 55: Comparação de tempo de execução da consulta  $Q_4$

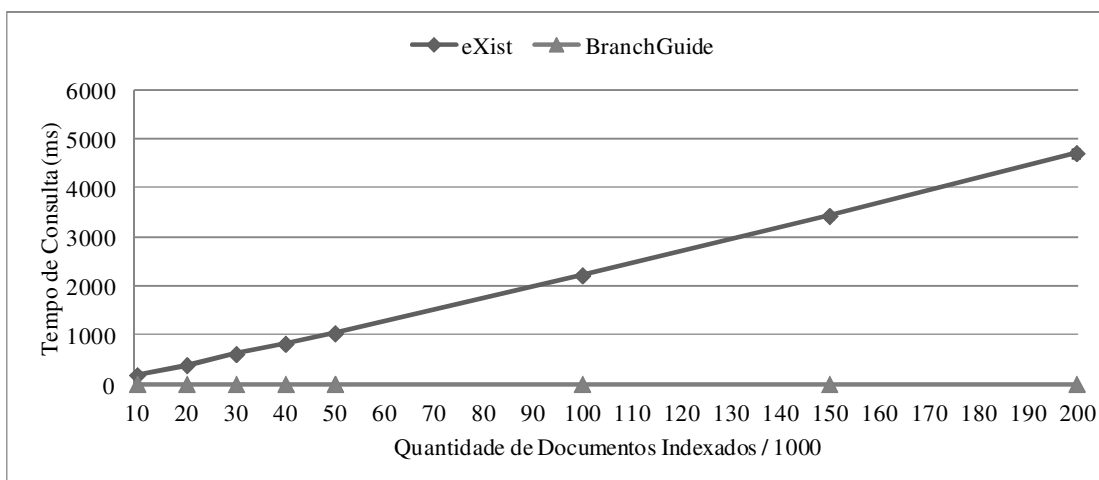


Figura 56: Comparação de tempo de execução da consulta  $Q_5$

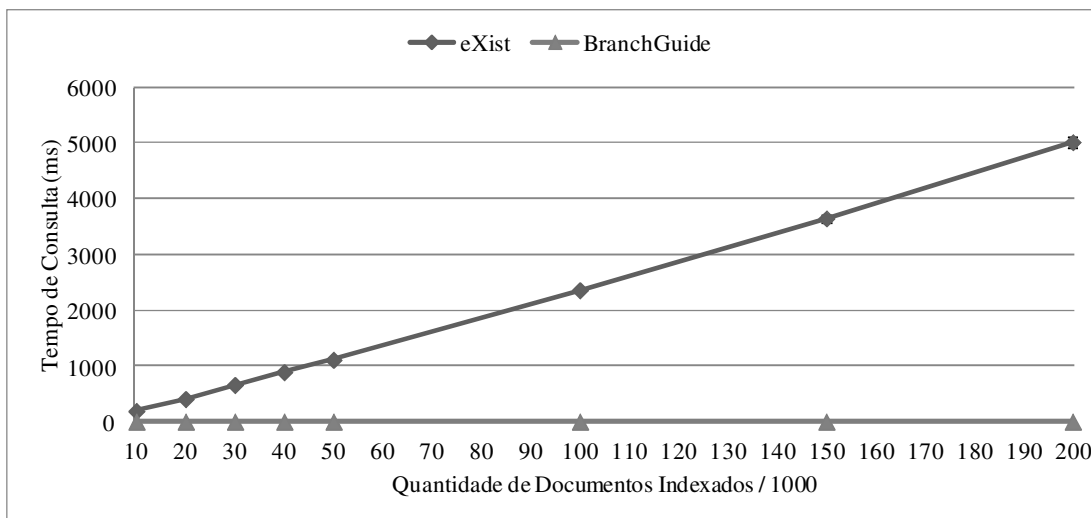


**Figura 57: Comparação de tempo de execução da consulta  $Q_6$**

Por fim, as Tabelas 19, 20, 21 e 22 detalham os valores de tempo de execução das consultas  $Q_3$ ,  $Q_4$ ,  $Q_5$ , e  $Q_6$  respectivamente.

**Tabela 19: Tempos de consulta para  $Q_3$**

| Quantidade de Documentos | eXist (ms) | Erro (ms): eXist | BranchGuide (ms) | Erro (ms): BranchGuide |
|--------------------------|------------|------------------|------------------|------------------------|
| 10000                    | 55,191     | 0,272            | 2,084            | 0,052                  |
| 20000                    | 111,726    | 0,491            | 4,348            | 0,108                  |
| 30000                    | 168,781    | 0,704            | 6,293            | 0,156                  |
| 40000                    | 225,820    | 1,162            | 8,536            | 0,213                  |
| 50000                    | 282,949    | 1,480            | 10,558           | 0,263                  |
| 100000                   | 555,187    | 2,729            | 23,432           | 0,585                  |
| 150000                   | 827,957    | 17,197           | 37,845           | 0,945                  |
| 200000                   | 1103,613   | 25,327           | 54,152           | 1,353                  |

**Tabela 20: Tempos de consulta para  $Q_4$**

| Quantidade de Documentos | eXist (ms) | Erro (ms): eXist | BranchGuide (ms) | Erro (ms): BranchGuide |
|--------------------------|------------|------------------|------------------|------------------------|
| 10000                    | 177,024    | 1,307            | 0,242            | 0,006                  |
| 20000                    | 378,713    | 0,709            | 0,214            | 0,003                  |
| 30000                    | 594,062    | 12,639           | 0,227            | 0,005                  |
| 40000                    | 804,195    | 14,061           | 0,228            | 0,005                  |
| 50000                    | 1004,139   | 15,514           | 0,224            | 0,001                  |
| 100000                   | 2126,799   | 42,171           | 0,236            | 0,005                  |
| 150000                   | 3263,262   | 70,202           | 0,230            | 0,001                  |
| 200000                   | 4470,152   | 101,496          | 0,235            | 0,005                  |

**Tabela 21: Tempos de consulta para  $Q_5$** 

| Quantidade de Documentos | eXist (ms) | Erro (ms): eXist | Técnica BranchGuide (ms) | Erro (ms): BranchGuide |
|--------------------------|------------|------------------|--------------------------|------------------------|
| 10000                    | 183,696    | 4,559            | 0,249                    | 0,001                  |
| 20000                    | 382,118    | 9,390            | 0,268                    | 0,001                  |
| 30000                    | 610,923    | 11,590           | 0,209                    | 0,001                  |
| 40000                    | 822,317    | 13,350           | 0,271                    | 0,001                  |
| 50000                    | 1032,135   | 21,015           | 0,225                    | 0,002                  |
| 100000                   | 2216,345   | 41,866           | 0,191                    | 0,004                  |
| 150000                   | 3422,617   | 64,304           | 0,291                    | 0,001                  |
| 200000                   | 4707,651   | 96,336           | 0,259                    | 0,001                  |

**Tabela 22: Tempos de consulta para  $Q_6$** 

| Quantidade de Documentos | eXist (ms) | Erro (ms): eXist | Técnica BranchGuide (ms) | Erro (ms): BranchGuide |
|--------------------------|------------|------------------|--------------------------|------------------------|
| 10000                    | 190,762    | 1,0741           | 0,284                    | 0,001                  |
| 20000                    | 402,761    | 9,869            | 0,313                    | 0,005                  |
| 30000                    | 656,287    | 11,015           | 0,286                    | 0,005                  |
| 40000                    | 887,787    | 19,617           | 0,244                    | 0,001                  |
| 50000                    | 1107,179   | 21,862           | 0,346                    | 0,002                  |
| 100000                   | 2354,725   | 42,817           | 0,348                    | 0,001                  |
| 150000                   | 3645,893   | 66,084           | 0,302                    | 0,001                  |
| 200000                   | 5013,008   | 97,287           | 0,257                    | 0,001                  |

Já os tempos de execução das consultas  $Q_7$ ,  $Q_8$  podem ser observados nos gráficos mostrados nas Figuras 58 e 59. De acordo com esses resultados, ao empregar a técnica BranchGuide, temos que o tempo de processamento da consulta  $Q_8$  que tem quatro operadores lógicos de conjunção é maior do que o tempo de processamento da consulta  $Q_7$  que tem dois operadores lógicos de conjunção; este aumento ocorre porque quanto maior a quantidade de operadores maior é a quantidade de consultas decompostas, aumentando assim o número de acessos ao índice juntamente com o tempo de combinação de resultados intermediários do processamento (isto é, a interseção de conjuntos de identificadores de documentos).

Mesmo com o custo deste processamento adicional, com a técnica BranchGuide, os tempos de execução de consultas são significativamente menores do que os tempos obtidos com a técnica de interseção estrutural adotada pelo eXist. Observa-se que, apesar da decomposição das várias consultas e das interseções necessárias para obter o resultado final, o tempo de execução total de uma consulta com operadores lógicos apresenta-se baixo em relação ao eXist pois o tempo de acesso das consultas decompostas (que são sub expressões com ramificação) é bastante baixo.

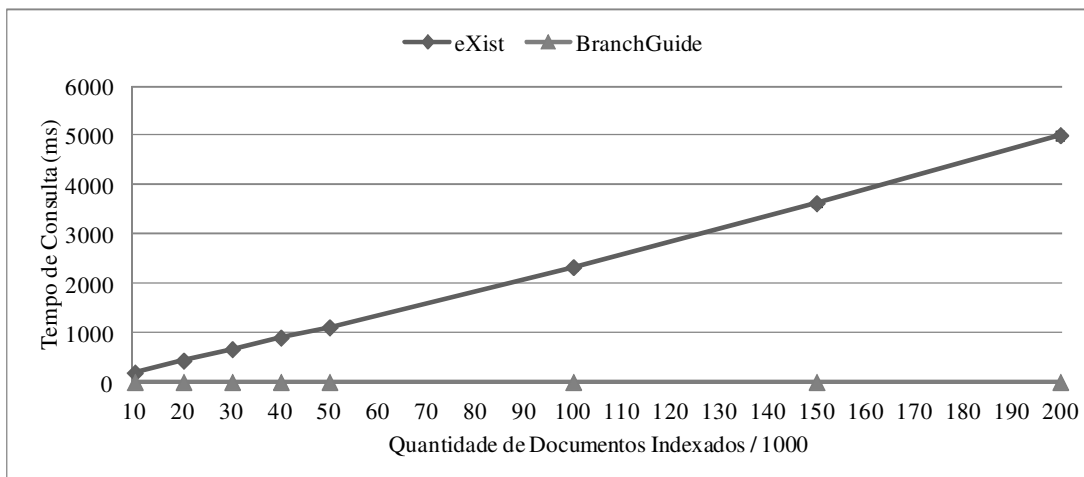


**Figura 58: Comparação de tempo de execução da consulta  $Q_7$**

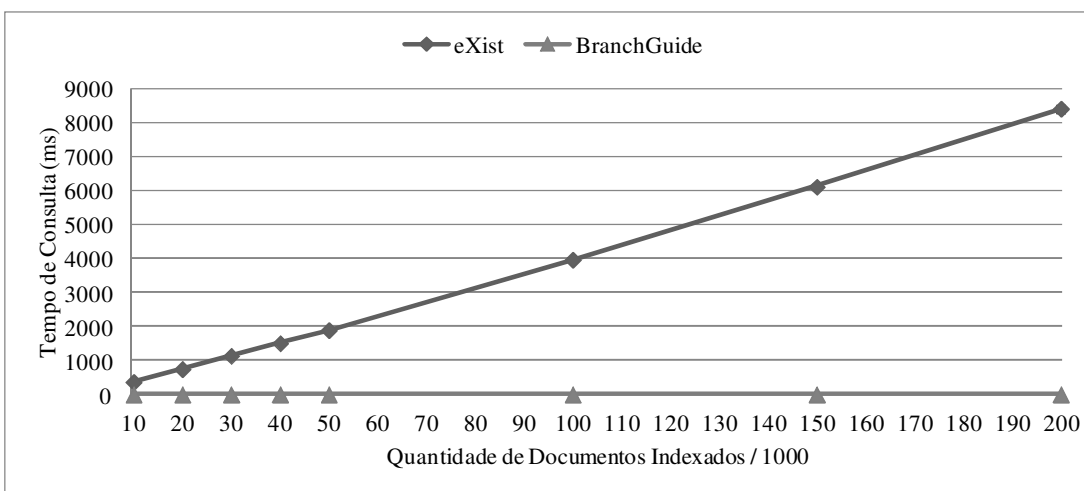


**Figura 59: Comparação de tempo de execução da consulta  $Q_8$**

Por fim, as Tabelas 23 e 24 detalham os valores de tempo de execução das consultas  $Q_7$  e  $Q_8$  respectivamente.

**Tabela 23: Tempos de consulta para  $Q_7$**

| Quantidade de Documentos | eXist (ms) | Erro (ms): eXist | BranchGuide (ms) | Erro (ms): BranchGuide |
|--------------------------|------------|------------------|------------------|------------------------|
| 10000                    | 191,734    | 1,058            | 0,831            | 0,021                  |
| 20000                    | 432,442    | 10,771           | 1,371            | 0,034                  |
| 30000                    | 662,634    | 15,576           | 0,851            | 0,011                  |
| 40000                    | 896,691    | 22,285           | 0,858            | 0,013                  |
| 50000                    | 1105,523   | 20,851           | 1,211            | 0,031                  |
| 100000                   | 2330,961   | 45,122           | 0,993            | 0,003                  |
| 150000                   | 3623,891   | 66,655           | 1,021            | 0,025                  |
| 200000                   | 5007,368   | 93,746           | 0,881            | 0,001                  |

Tabela 24: Tempos de consulta para  $Q_8$ 

| Quantidade de Documentos | eXist (ms) | Erro (ms): eXist | BranchGuide (ms) | Erro (ms): BranchGuide |
|--------------------------|------------|------------------|------------------|------------------------|
| 10000                    | 363,735    | 7,911            | 1,646            | 0,041                  |
| 20000                    | 744,646    | 15,164           | 2,311            | 0,057                  |
| 30000                    | 1129,865   | 19,704           | 2,274            | 0,041                  |
| 40000                    | 1505,716   | 18,796           | 2,136            | 0,052                  |
| 50000                    | 1887,094   | 26,201           | 1,671            | 0,041                  |
| 100000                   | 3959,619   | 48,747           | 1,848            | 0,045                  |
| 150000                   | 6105,249   | 78,086           | 2,396            | 0,059                  |
| 200000                   | 8399,894   | 102,291          | 1,915            | 0,047                  |

### 6.4.3 Tempo de Indexação

O gráfico mostrado na Figura 60 apresenta o tempo de indexação da amostra X-ARM utilizando a técnica BranchGuide e o eXist. Os tempos de indexação da técnica BranchGuide variam no máximo em aproximadamente 78 minutos e no mínimo por aproximadamente 4 minutos em relação com os tempos obtidos no eXist.

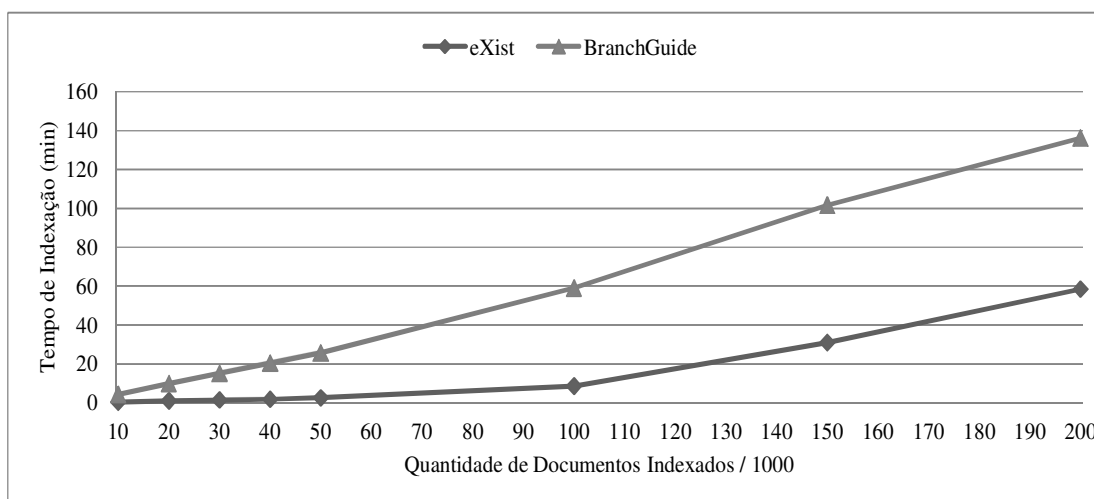


Figura 60: Tempo de indexação: X-ARM

O comportamento esperado contrasta com os resultados esperados para a outra amostra indexada DBLP (resultados discutidos anteriormente na Seção 6.3.3). Temos que o eXist extrai nomes de elementos e atributos dos documentos enquanto que a técnica BranchGuide extrai caminhos com ramificação para indexação. No caso da amostra X-ARM, o número de caminhos extraídos é bastante grande (conforme apresentado na Tabela 13), e consequentemente, maior do que o número de nomes da fonte de dados. Como a amostra X-ARM gera uma grande quantidade de ocorrências de caminhos com ramificação, então o tempo necessário para a indexação da amostra com a técnica BranchGuide é geralmente maior do que com o eXist.

Apesar dos tempos obtidos com a técnica BranchGuide serem geralmente maiores do que os tempos obtidos com o eXist, esta diferença entre os tempos não inviabiliza a adoção da técnica BranchGuide. Por exemplo, uma amostra com *200.000 mil* documentos levou cerca de *140 minutos* para ser indexada.

## **6.5 Considerações Finais**

A avaliação experimental apresentada neste capítulo não engloba a investigação qualitativa do método de recuperação da técnica BranchGuide. Tal investigação qualitativa tem como objetivo medir o grau de dificuldade para o usuário construir consultas utilizando expressões com ramificação do padrão suportado pela técnica bem como verificar com que porcentagem os resultados de pesquisa retornados satisfazem as necessidades iniciais do usuário.

Outro segundo ponto de investigação diz respeito à avaliação da qualidade do método de redução de replicação de informações do grafo de índice gerado pela técnica BranchGuide. Isto é, verificar experimentalmente qual é a porcentagem total de vértices duplicados (do grafo de índice) que não são armazenados e determinar qual é a correlação desta redução com o formato (isto é, a disposição dos elementos/atributos) dos documentos de uma amostra. O principal objetivo desta investigação é delinear o comportamento de redução de tamanho de índice em função de diferentes configurações de formato das amostras.

As investigações abertas citadas estão fora do escopo deste trabalho e compreendem um conjunto de estudos que deverão ser desenvolvidos em trabalhos futuros.

## 7 Considerações Finais

A técnica BranchGuide adota um padrão de seleção para uma classe de expressões com ramificação. O principal objetivo é que este conjunto de expressões com ramificação seja indexado sem perda de precisão. Neste caso, para processar consultas, não é necessário empregar a decomposição em nomes de atributos/elementos juntamente com o uso de interseção estrutural. Vale ressaltar que o conjunto de caminhos indexados possibilita processar expressões com semântica mais abrangente do que na indexação de caminhos simples, aliás, incluindo os caminhos simples como um subconjunto do total de caminhos indexados.

A fim de extrair o conjunto de caminhos com ramificação suportado a partir de uma fonte de dados XML, nesta dissertação é especificado um padrão de seleção de caminhos que tem como principal requisito capturar importantes relações entre os elementos/atributos da fonte de dados que seriam perdidas caso fossem empregadas técnicas de indexação de caminhos simples. A extração de caminhos com ramificação que conformam com o padrão de seleção proposto segue a execução dos passos de processamento discutidos no Capítulo 3. Em resumo, estes passos geram um grafo de índice que contém explicitamente as ramificações que são indexadas a partir da fonte de dados. Este processo de extração é projetado de modo que não se perca informações relacionadas aos caminhos com ramificação indexados, assim, tornando a técnica BranchGuide precisa.

É importante ressaltar que após processo de extração de caminhos, a técnica BranchGuide gera uma quantidade não linear de expressões com ramificação a serem armazenadas no índice. Desta forma, nesta dissertação é definido um projeto de armazenamento que reajusta o modo como que informações do índice são armazenadas (discutido no Capítulo 4). Isto é feito com o objetivo de minimizar os requisitos de armazenamento, e assim, prover tamanhos de índice menores do que os gerados através de técnicas de indexação baseadas em interseção estrutural.

Ao explorar os grafos de índice gerados pela técnica BranchGuide, ocorre-se a observação de que existem informações replicadas dispersas no grafo de índice, assim, neste trabalho é proposto um método de ajuste de armazenamento que minimiza a ocorrência de informações duplicadas. Os ajustes adotados no armazenamento do índice provêm uma significativa redução dos tamanhos dos arquivos de índice gerados. Como resultado, os tamanhos de índice gerados pela técnica BranchGuide são próximos ou maiores ao tamanho da amostra indexada, apesar disso, mesmo nos casos em que são maiores, os tamanhos de índice são geralmente menores do que os tamanhos de índice gerados por técnicas de interseção estrutural. É importante ressaltar que mesmo com a redução dos tamanhos dos arquivos de índice, os ajustes empregados não degradam a precisão do índice, bem como não degradam o tempo de execução de consultas.

Em termos de implementação, é adotada a estrutura de dados de índice invertido (discutida no Capítulo 5), a qual oferece um mecanismo eficiente para a descoberta dos resultados. Os índices gerados pela técnica BranchGuide recebem como entrada um caminho com ramificação (ou simples) que pertença a classe de consultas suportada e, com isto, executa-se um único acesso ao índice trazendo de uma única vez como resultado os identificadores de todos os documentos nos quais existem ocorrências do caminho pesquisado.

Com o objetivo de tratar o problema de replicação de informações armazenadas no índice, no projeto de implementação do índice (discutida no Capítulo 5), uma função de *hash* perfeito exclusivamente modelada mapeia os nomes de expressões de caminhos para códigos inteiros numéricos. Estes códigos inteiros quando ordenados respeitam os requisitos observados para o armazenamento com a redução de replicação. Com isto, determinadas chaves de índice que têm o mesmo resultado duplicado são armazenadas sequencialmente de forma que não é necessário armazenar todos os resultados de pesquisa deste conjunto, mas somente o resultado de pesquisa associado à chave que inicia a sequência. Conforme evidenciado experimentalmente (Capítulo 6), o projeto adotado gera tamanhos de índice menores do que os gerados por uma técnica de interseção estrutural mesmo em situações no qual a amostra possui muitos atributos, situação em que é gerada uma grande quantidade de caminhos com ramificação a indexar.

As subseções a seguir listam as principais contribuições da técnica BranchGuide. Também são enumerados os benefícios da técnica BranchGuide e quais as vantagens de se adotar a técnica BranchGuide em um sistema de recuperação de informação para dados semi-

estruturados. Por fim são listadas e discutidas algumas limitações da técnica proposta e como isso pode se tornar motivação para novos estudos e desenvolvimento de trabalhos futuros.

## 7.1 Contribuições

Nesta dissertação, uma técnica de indexação baseada em caminhos para dados semi-estruturados (denominada BranchGuide) é proposta e concebida. Dentre as principais contribuições originadas a partir deste trabalho, temos a proposição de uma nova técnica de indexação que provê suporte para uma classe de consultas com ramificação que possibilita processar expressões de consultas mais expressivas do que as expressões de caminhos simples, suportadas por índices tal como o Dataguide [Goldman97]. Além disso, a técnica BranchGuide não sofre de perda de precisão (isto é, não retorna falsos positivos) ao processar as expressões de consulta suportadas. Como evidenciado por uma avaliação experimental, em contraste com outras propostas baseadas em interseção estrutural (ex: a técnica de indexação empregada no sistema eXist [Meier02]), a técnica BranchGuide não sofre do crescimento indesejado dos tempos de processamento de consultas e tamanhos de índice.

Como inicialmente reportado por Kaushik [Kaushik04], o emprego de operações de interseção estrutural implica em altos custos no processamento de consultas, logo, quanto menor é quantidade de operações de interseção estrutural executadas então maior é o aprimoramento obtido no tempo de processamento de consultas. Como a técnica BranchGuide não emprega operações de interseção estrutural, é esperado que o aprimoramento para o tempo de processamento de consultas suportadas pela técnica BranchGuide deva ser ótimo.

Kaushik [Kaushik04] propõe uma combinação do uso de índices de indexação de caminhos simples juntamente com técnicas de interseção estrutural. No processamento de consultas com ramificação, ao invés da decomposição de expressões em termos ocorre a decomposição das consultas com ramificação em um conjunto de consultas de caminho simples (cada ramificação da consulta original gera um caminho simples a ser pesquisado). Assim, a quantidade de operações de interseção é menor do que no caso de decomposição em termos. Por meio disto, o trabalho de Kaushik consegue obter um aprimoramento em relação ao tempo de processamento de consultas. A técnica BranchGuide pode ser vista como um passo à frente ao trabalho de Kaushik já que na técnica BranchGuide o tempo de processamento de consultas é aprimorado mais ainda. Ao invés de decompor expressões com ramificação em caminhos simples, temos que as expressões com ramificação suportadas não

precisam ser decompostas, e assim, são diretamente pesquisadas sem o uso de operações de interseção estrutural.

Como evidenciado através de experimentos, a fim de se obter um aprimoramento sob o tempo de processamento de consultas, indexar uma grande quantidade de caminhos com ramificação é uma melhor direção a seguir do que usar operações de interseção estrutural para processar caminhos com ramificação. Apesar disso, vale ressaltar que a indexação de uma grande quantidade de caminhos com ramificação implica em uma grande quantidade de informações a ser armazenada. A fim de reduzir a complexidade de armazenamento, a técnica BranchGuide faz uso de um projeto de armazenamento de índice que diminui a replicação de informação armazenada mas sem empregar mecanismos que causem a perda de informação como efeito colateral.

Também evidenciado por experimentos, os ajustes adotados no armazenamento não degradam o tempo de processamento de consultas, mas ao contrário, o tempo para pesquisar qualquer caminho suportado em índice é constante. Como contribuição, a proposição e concepção da técnica BranchGuide pode ser vista como um modelo de projeto de índice sem perda de informação que ao mesmo tempo consegue prover um mecanismo escalável para o processamento de consultas.

Além disso, pelo fato da técnica BranchGuide prover uma solução eficiente do ponto de vista de processamento de consultas, o que é um requisito básico para os usuários de sistemas de recuperação, é importante ressaltar que, na prática, esta propriedade é fundamental para potencializar o emprego da técnica em sistemas de recuperação de informação para dados semi-estruturados.

Por fim, vale ressaltar que apesar da técnica BranchGuide ser projetada para indexar diretamente uma classe de caminhos com ramificação mais expressiva do que a classe de caminhos simples, esta não é restrita somente a este subconjunto de caminhos com ramificação. A técnica BranchGuide é modelada para que informações auxiliares possam ser adicionadas no índice. Logo, a técnica BranchGuide define um ponto de extensão que pode ser explorado na construção de diferentes e novas técnicas de indexação baseadas em índices BranchGuide. Por exemplo, além dos caminhos com ramificação suportados, através do armazenamento de informação auxiliares temos que a técnica BranchGuide consegue tratar operadores lógicos que definem restrições entre atributos XML.

## 7.2 Benefícios

Nesta seção são enumerados um conjunto de fatores que trazem benefícios com a adoção da técnica BranchGuide para a indexação de dados semi-estruturados em sistemas de busca. Cada um dos fatores e os respectivos benefícios gerados por são apresentados e discutidos a seguir.

**Técnica Projetada para Sistemas de Busca de Larga Escala.** Sistemas de busca requerem eficiente processamento de consultas bem como métodos de armazenamento apropriados para uma fonte de dados de larga escala, caso contrário, a indexação de uma grande quantidade de dados pode inviabilizar a utilização do sistema de busca devido à lentidão para processar requisições de consulta. Além disso, um método de armazenamento adequado possibilita gerar índices mais compactos e, assim, reduzir o custo para o armazenamento do índice.

Conforme demonstrado experimentalmente, a técnica BranchGuide gera excelentes tempos de processamento de consulta e tamanhos de índice menores em comparação com uma técnica de interseção estrutural. Logo, isto torna mais adequado o emprego da técnica BranchGuide em sistemas de busca de documentos semi-estruturados em detrimento de técnicas de interseção estrutural.

**Indexação de Caminhos com Ramificação.** As técnicas baseadas em caminhos simples não suportam o processamento de consultas com ramificação. Mesmo se uma consulta com ramificação for decomposta em caminhos simples para assim obter o resultado final através da interseção de resultados intermediários, temos que os resultados finais podem não satisfazer a consulta com ramificação (retornando falsos positivos). Isto ocorre basicamente porque as informações que permitem expressar dependências entre os elementos/atributos em caminhos com ramificação não são indexadas com o objetivo de reduzir o custo de armazenamento do índice.

Em contraste, a técnica BranchGuide é modelada para indexar uma quantidade expressiva de caminhos com ramificação além dos caminhos simples. O principal benefício é que sistemas de recuperação de informação podem adotar a técnica BranchGuide em seu motor de indexação para assim prover funcionalidades de consulta mais expressivas, isto é, consultas que levem em conta uma mais significativa quantidade de relações entre os elementos/atributos pesquisados. Isto tende a aperfeiçoar o nível de expressividade das necessidades de informação dos usuários em sistemas de recuperação de informação.

**Indexação Exata.** A técnica BranchGuide indexa uma classe de caminhos com ramificação de forma exata. Isto é, no processamento de consultas, para esta classe de consultas com ramificação suportada nunca é retornado um documento que não satisfaz a ocorrência do caminho pesquisado. Algumas outras técnicas (tal como a técnica ViST [Wang03]) fazem uso da redução de precisão para obter uma redução da complexidade de armazenamento do índice. Já a técnica BranchGuide adota um método diferente de tratamento de redução de complexidade de armazenamento que não emprega mecanismos que geram imprecisão. Com isto, espera-se que sistemas de recuperação para dados semi-estruturados que adotem a técnica BranchGuide obtenham um aperfeiçoamento em relação à precisão total do sistema de recuperação.

**Controle de Similaridade: Suporte a Operadores Lógicos.** A técnica BranchGuide possibilita o tratamento de operadores lógicos sob os valores de atributos consultados. Isto é, a técnica BranchGuide adota o modelo de recuperação de informação baseado em operadores lógicos. Os operadores lógicos permitem definir restrições de pesquisa com diferentes níveis de similaridade, de forma que é possível expressar consultas com diferentes níveis de rigorosidade (restrições mais ou menos estritas). Do ponto de vista da aplicação em sistemas de busca, temos que o controle de similaridade é uma funcionalidade importante já que o usuário do sistema consegue definir o nível de similaridade dos resultados descobertos pelo sistema em relação ao que o usuário realmente deseja procurar. Em algumas situações, documentos que satisfazem parcialmente uma necessidade de informação podem ser úteis no processo de descoberta de novos documentos.

Especificamente, a técnica BranchGuide é projetada de forma a prover um modelo de recuperação de informação com similaridade baseado em operadores lógicos, o que é um recurso bastante importante em sistemas de recuperação de informação.

**Indexação em Memória Secundária.** Além de prover excelentes resultados para o processamento de consultas e geração de índices de baixo custo, é importante também destacar que o projeto do mecanismo de indexação da técnica BranchGuide leva em conta requisitos relacionados com a escalabilidade do processo de indexação. O mecanismo de indexação da técnica BranchGuide adota um algoritmo de indexação em que a quantidade máxima de dados a indexar pode ser maior do que o espaço disponível em memória principal, para isso utiliza-se de memória secundária para armazenamento temporário das informações parciais indexadas. Neste sentido, esta funcionalidade provê uma maior escalabilidade do processo de indexação, permitindo assim que a quantidade de informações que podem ser

indexadas possa aumentar sem problemas. Desta forma, temos que a técnica BranchGuide pode ser adotada (em termos práticos) em sistemas de indexação para uma volumosa quantidade de dados semi-estruturados.

### 7.3 Limitações

Apesar das contribuições e benefícios trazidos pela técnica BranchGuide, vale ressaltar que a técnica sofre de limitações que são enumeradas a seguir.

**Impacto de Modificação da Fonte de Dados.** O algoritmo de indexação empregado na técnica BranchGuide é do tipo estática. Isto é, na indexação estática temos que um índice já computado não pode ser alterado após ter sido gerado como saída do processo de indexação. Logo, se a fonte de dados for alterada, a fim de obter o índice que reflita esta alteração é necessário reindexar a fonte de dados por completo novamente. Quando a indexação estática é empregada em geral são executadas reindexações de forma periódica.

Neste sentido, ao adotar a técnica BranchGuide, qualquer modificação da fonte de dados não implica diretamente na alteração do índice, de forma que a técnica BranchGuide deve ser empregada em sistemas de busca que empregam um esquema de agendamento de indexações (como por exemplo, a reindexação periódica). É importante ressaltar que a indexação periódica é uma boa alternativa se o número de mudanças da fonte de dados no decorrer do tempo é pequena e ter atraso na inserção de novos documentos ao índice é aceitável. Além disso, recursos de armazenamento devem estar disponíveis para construir um novo índice enquanto o anterior ainda continua disponível para pesquisa [Manning08]. Logo, a técnica BranchGuide oferece limitações para os casos em que tais quesitos não são satisfeitos.

**Análise Teórica da Complexidade de Armazenamento.** De acordo com os resultados apresentados em [Goldman97], um índice Dataguide em situação de pior caso tem complexidade de armazenamento linear. Já na técnica BranchGuide, pode-se constatar através de observação que para uma amostra que contenha pelo menos um atributo, devido à transformação que ocorre no processo de extração do grafo de índice (conforme discutido na Seção 3.2), temos que o grafo de índice gerado pela técnica BranchGuide se torna maior do que o grafo de índice gerado em um Dataguide. Apesar disto, ainda não é determinada a função de complexidade de armazenamento de pior caso de um índice BranchGuide.

É importante ressaltar que a determinação da complexidade de pior caso do armazenamento de índices gerados pela técnica BranchGuide em função do número de atributos/elementos da fonte de dados não faz parte do escopo deste trabalho. Isto acaba caracterizando certa limitação do trabalho. Ainda é um problema aberto determinar se o grafo de índice gerado pela técnica BranchGuide tem complexidade exponencial ou não. Esta é uma questão importante a se investigar, pois algoritmos com complexidade exponencial têm problemas de escalabilidade, isto é, não executam bem em condições nas quais as entradas do algoritmo têm um tamanho de ordem infinita. Do ponto de vista experimental, problemas de escalabilidade não foram reportados nos experimentos realizados com a técnica BranchGuide, mesmo assim, a determinação teórica dos custos de armazenamento permitiria estabelecer com mais rigor que os resultados gerados pela indexação via BranchGuide são teoricamente escaláveis para qualquer tipo de entrada. Além disso, a determinação da complexidade de armazenamento permitiria comparar teoricamente o algoritmo BranchGuide com um catálogo de algoritmos de indexação já existentes.

## 7.4 Trabalhos Futuros

Como direção de trabalho futuro tem-se a investigação qualitativa do método de recuperação da técnica BranchGuide. Isto incluiria medir o grau de dificuldade para o usuário construir consultas utilizando expressões com ramificação do padrão suportado pela técnica bem como utilizar operadores para controlar o grau de similaridade para os resultados obtidos. Além disso, trabalhos futuros incluem a construção de um modelo qualitativo de medição da satisfação do usuário quando empregado a técnica BranchGuide para recuperar documentos semi-estruturados. Este modelo de medição deve determinar valores percentuais de satisfação médio do usuário que utiliza um sistema de recuperação de informação o qual emprega a técnica BranchGuide para a descoberta de documentos semi-estruturados.

Um segundo ponto de investigação futuro diz respeito à qualidade do método de redução de replicação de informações no grafo de índice gerado pela técnica BranchGuide. O principal objetivo desta investigação é determinar fatores percentuais de redução do tamanho do arquivo índice em função de diferentes características das amostras, as quais poderiam ser parametrizadas (como por exemplo, a quantidade de atributos, o número de elementos, bem como diferentes relações entre elementos dos documentos). Na condução desta investigação o primeiro passo é a modelagem de uma expressão matemática que determina o fator de redução do tamanho do índice em função das características da amostra que podem ser

parametrizadas. Esta função tem o potencial de demonstrar teoricamente a adaptabilidade da técnica proposta para um conjunto genérico de amostras.

Além disso, como esforço futuro, é importante verificar se a função que determina o fator de redução de duplicação se confirma em situação experimental. Como um conjunto genérico (e infinito) de formatos de amostras não pode ser completamente experimentado na prática, um conjunto de amostras geradas através de um conjunto de probabilidades estaticamente determinadas deve ser indexado com a técnica BranchGuide. O objetivo final é analisar o comportamento real de redução de tamanho com o projeto de algoritmo de indexação empregado na técnica. Estes resultados poderiam firmar uma aplicabilidade geral da técnica em diferentes cenários nos quais diversos tipos de amostras são indexados.

Neste trabalho é adotado um modelo de recuperação de informação baseado em operadores lógicos (modelo *booleano* de recuperação), em que o usuário controla a similaridade dos resultados através do uso de operadores lógicos. Como investigação futura, outros modelos de recuperação podem ser utilizados. Por exemplo, ao adotar um modelo de recuperação baseado em frequências temos que informações sobre a frequência de ocorrência dos caminhos poderia ser armazenada em índice. Tais informações de frequência determinariam se um documento satisfaz melhor uma consulta tomando a heurística em que um documento terá uma melhor probabilidade de satisfazer a consulta se um caminho buscado ocorre com maior frequência neste documento e com menor frequência em todos os outros da coleção.

Por fim, espera-se que os resultados experimentais obtidos no desenvolvimento técnica BranchGuide incentivem o desenvolvimento de novas técnicas de indexação baseadas em caminhos, bem como, tipos mistos de índices os quais empreguem índices BranchGuide juntamente com técnicas de indexação de outras categorias. Além disso, pelo fato de viabilizar a recuperação de documentos semi-estruturados em um cenário de larga-escala, provendo eficiência em função dos requisitos de precisão, tempo de recuperação e armazenamento, espera-se que a técnica BranchGuide fomente a concepção, projeto e desenvolvimento de novos sistemas de busca na Web que levem em conta aspectos estruturais (semânticos) dos documentos. Sendo assim, o desenvolvimento da técnica de indexação BranchGuide torna-se um passo primário na proposição de novos sistemas de busca de documentos semi-estruturados na Web.

## Referências

- [Catania05] Catania, B.; Maddalena, A.; Vakali, A., "*XML document indexes: a classification*," IEEE Internet Computing , vol.9, no.5, pp. 64-71, Sept.-Oct. 2005.
- [Cormen01] T. H. Cormen, C. E. Leiserson, R. L. Rivest (2001). "*Introduction to Algorithms*," The MIT Press.
- [Goldman97] R. Goldman and J. Widom. DataGuides: "*Enabling Query Formulation and Optimization in Semistructured Databases*". In Proceedings of the Twenty-Third International Conference on Very Large Data Bases (VLDB), 1997.
- [Gou07] G. Gou and R. Chirkova, 2007. "*Efficiently Querying Large XML Data Repositories: A Survey*". IEEE Transactions on Knowledge and Data Engineering, 2007.
- [Harold01] Harold. E. "*XML Bible*", Gold John Wiley & Sons, Inc, 2001.
- [Jain91] Raj Jain, "*The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*," Wiley-Interscience, New York, NY, April 1991.
- [Jiang05] Wei Wang, Haifeng Jiang, Hongzhi Wang, Xuemin Lin, Hongjun Lu, and Jianzhong Li. 2005. "*Efficient processing of XML path queries using the disk-based F&B Index*". In Proceedings of the 31st international conference on Very large data bases (VLDB '05). VLDB Endowment 145-156.
- [Kamps06] J. Kamps, M. Marx, M. Rijke, d, and B. Sigurbjörnsson. "*Articulating Information Needs in XML Query Languages*". ACM Transactions on Information Systems 24, 2006.
- [Kaushik04] Raghav Kaushik , Rajasekar Krishnamurthy , Jeffrey F. Naughton , Raghu Ramakrishnan, "*On the integration of structure indexes and inverted lists*",

Proceedings of the 2004 ACM SIGMOD international conference on Management of data, June 13-18, 2004, Paris, France.

- [Ley09] M. Ley, “*DBLP Database Web Site*”, <http://www.informatik.uni-trier.de/~ley/db>, 2009.
- [Li01] Quanzhong Li , Bongki Moon, “*Indexing and Querying XML Data for Regular Path Expressions*”, Proceedings of the 27th International Conference on Very Large Data Bases, p.361-370, September 11-14, 2001.
- [Manning08] Manning, C.; Raghavan, P.; Schütze, H. (2008) “*Introduction to Information Retrieval*”. Cambridge University Press.
- [Meier02] Wolfgang Meier, “*eXist: An Open Source Native XML Database*”, Revised Papers from the NODe 2002 Web and Database-Related Workshops on Web, Web-Services, and Database Systems, p.169-183, October 07-10, 2002.
- [Milo99] Tova Milo and Dan Suciu. 1999. “*Index Structures for Path Expressions*”. In Proceedings of the 7th International Conference on Database Theory (ICDT '99), Catriel Beeri and Peter Buneman (Eds.). Springer-Verlag, London, UK, 277-295.
- [Sinha11] Sinha, Nishit K., “*Demystifying Number System: (Practical Concepts and Their Applications) for the CAT and Other MBA Exams*”, Pearson Education India, 2011.
- [Rijsbergen79] C. J. Van Rijsbergen. 1979. “*Information Retrieval*” (2nd ed.). Butterworth-Heinemann, Newton, MA, USA.
- [Saltenis04] S. Saltenis, “*External-Memory Sorting*”, Lecture Notes, <http://www.cs.aau.dk/~simas/aalg04/lectures.html>, 2004.
- [Schuenck06] Schuenck, M. (2006) “*X-ARM: Um Modelo de Representação de Artefatos de Software*”. Dissertação de Mestrado, DIMAp-UFRN, Natal-RN.
- [Timo04] Timo Böhme, Erhard Rahm, “*Supporting Efficient Streaming and Insertion of XML Data in RDBMS*”, Zohra Bellehsène, Peter J. McBrien (Ed.), Third International Workshop on Data Integration over the Web, Riga, Latvia, June 2004.

- [Trotman05] Trotman A.; Sigurbjörnsson B. "*Narrowed extended XPath I (NEXI)*". In *Advances in XML Information Retrieval*, 2005.
- [W3C07] W3C, "*Extensible Markup Language*" (XML), <http://www.w3.org/XML/>, 2007.
- [W3C99] W3C, "*XML Path Language (XPath)*", <http://www.w3.org/TR/xpath>, 1999.
- [Wang03] Wang H.; Park S.; Fan W. and Yu S. "*ViST: a Dynamic Index Method for Querying XML Data by Tree Structures*". In *Proc. of the 2003 ACM SIGMOD international Conference on Management of Data*, 2003.
- [Wang05] Haixun Wang and Xiaofeng Meng. 2005. "*On the Sequencing of Tree Structures for XML Indexing*". In *Proceedings of the 21st International Conference on Data Engineering (ICDE '05)*. IEEE Computer Society, Washington, DC, USA, 372-383.
- [Weigel02] Weigel, F. (2002) "*A Survey of Indexing Techniques for Semistructured Documents*". Project Thesis, Institute for Computer Science, University of Munich, 2002.

# Apêndice

## A.1 Notação Adotada

Neste apêndice a notação utilizada nesta dissertação para a representação visual de fontes e índices de dados semi-estruturados é apresentada. Inicialmente é importante ressaltar que nos modelos de dados semi-estruturados os valores de dados são armazenados juntamente com metadados relacionados. Sendo assim, é necessária a definição de uma notação que possibilite diferenciar os dados da informação semântica associada. A fim de adotar um método para apresentação de informações, nesta seção é definida uma notação baseada em grafos que permite mapear e diferenciar a estrutura (semântica) e valores de dados de documentos XML. Esta notação é baseada na notação original definida por Weigel [Weigel02]. Para o desenvolvimento deste trabalho, a diferença entre elementos e atributos XML é bem definida, neste sentido, é adotada uma notação baseada em grafos para diferenciação de tais conceitos. A seguir, tal notação é detalhada.

Para representação de elementos, como mostrado na Figura 61, é utilizado um vértice com uma aresta incidente. A aresta incidente é rotulada com o nome do elemento (indicado ao lado da aresta), já o vértice pode mapear informações arbitrárias dependentes de uma definição específica.

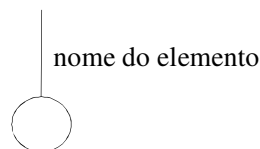


**Figura 61: Representação de um elemento**

Já para representar atributos, como mostrado na Figura 62, é utilizado um vértice cinza com uma aresta incidente. A aresta incidente é rotulada com o nome do atributo (indicado ao lado da aresta). Já o vértice pode mapear informações arbitrárias dependentes de uma definição específica. Além disso, um vértice de atributo tem um valor de dado associado, neste caso, tal valor associado é indicado ao lado do vértice. Vale ressaltar que, para facilitar a

diferenciação entre atributos e elementos, os vértices de elemento são mostrados com cor branca e os vértices de atributo com cor cinza.

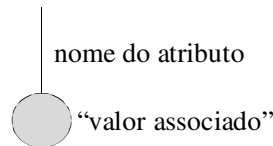


**Figura 62: Representação de um atributo**

Como o XML é um modelo de representação de dados hierárquico, podemos representar qualquer documento XML através de um grafo (especificamente, uma árvore, já que no modelo XML não é possível definir ciclos). Por exemplo, na Figura 63, é mostrado como tal hierarquia é apresentada utilizando a notação adotada. Para isto, devemos considerar que todas as arestas são unidirecionais e incidem nos vértices de elemento sempre no sentido *pai*→*filho*. Quando conectamos ao menos dois elementos é definida uma hierarquia estrutural dos dados. Na Figura 63, o vértice de elemento denominado por *elemento pai* é pai do elemento denominado *elemento filho*.

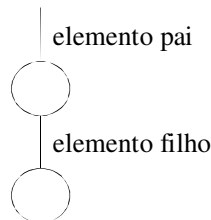


**Figura 63: Hierarquia entre elementos**

Além disso, como citado anteriormente, existe uma relação entre um dado elemento e seus respectivos atributos. Esta relação é representada de forma análoga à hierarquia entre elementos, mas neste caso, um vértice de elemento é conectado a um vértice de atributo, tal como mostrado na Figura 64. Vale ressaltar que no XML, um atributo descreve uma propriedade particular de um elemento, de forma que não existem elementos ou atributos filhos de um atributo, sendo assim, vértices de atributos são sempre nós folha de um documento representado utilizando a notação baseada em grafo.

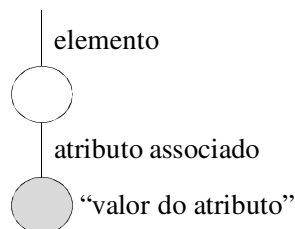


**Figura 64: Atributo de um elemento**

A notação adotada nesta dissertação é uma extensão da notação definida por Weigel [Weigel02]. Na notação de Weigel, os vértices mapeiam um identificador unívoco do elemento ou atributo em relação a uma fonte de dados composta por vários documentos. Em contrapartida, neste trabalho, os vértices mapeiam diferentes informações que são discutidas detalhadamente na Seção 3.2. De forma resumida, estas informações incluem identificadores unívocos dos documentos em relação à fonte de dados, e, além destes, no caso de vértices de atributos, também são mapeados códigos identificadores de atributos em relação ao documento que está contido.

Para fins de exemplificação, a Figura 65a mostra um documento XML identificado por *Documento 1*. Este documento descreve os livros publicados por uma editora. No documento, o elemento *autor* é filho do elemento *livro* que por sua vez é filho do elemento *editora*. Esta relação de hierarquia de três níveis é denotada na Figura 65b. Além disso, na Figura 65b, os elementos *editora*, *livro* e *autor* têm vértices de atributos (ressaltados de cor cinza) conectados aos respectivos elementos, bem como o valor de cada atributo é denotado ao lado do respectivo vértice de atributo.

Por fim, neste exemplo, o valor mapeado por cada vértice é definido como o identificador do documento onde existe o respectivo elemento ou atributo. Além disso, os valores mapeados são mostrados no interior de cada vértice, desta forma, na Figura 65b, dentro de cada vértice são ilustrados os valores *1* relativos à ocorrência dos elementos/atributos no *Documento 1*.

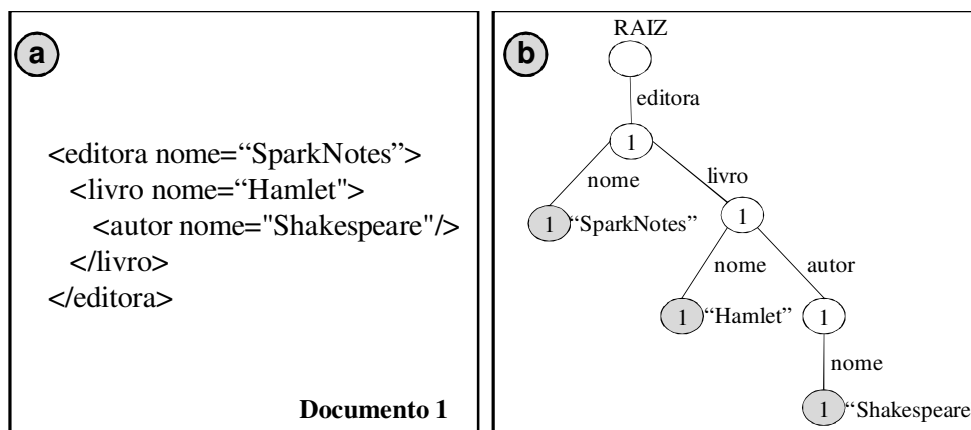


**Figura 65: Exemplo de Documento XML e grafo correspondente**

## A.2 Linguagem XPath

A linguagem XPath possibilita localizar partes de documentos XML. Em geral, XPath permite expressar consultas de caminhos simples com predicados, como por exemplo, a expressão `/editora/livro[@nome="Hamlet"]` (ilustrada na Figura 66a). Esta expressão seleciona nós de elemento *livro* tal que tenha o atributo *nome* com valor “Hamlet”.

Além disso, a linguagem XPath permite especificar consultas mais complexas, tal como, consultas com ramificação. Por exemplo, a expressão `/editora/livro/nome[@nome="Hamlet"]/autor[@nome="Shakespeare"]` (ilustrada na Figura 66b) seleciona nós de elemento *autor* com nome “Shakespeare”, mas, além disso, o autor deve ter escrito um *livro* de nome “Hamlet”. Nas ilustrações da Figura 66a e Figura 66b o nó alvo de pesquisa está destacado através de preenchimento pontilhado.

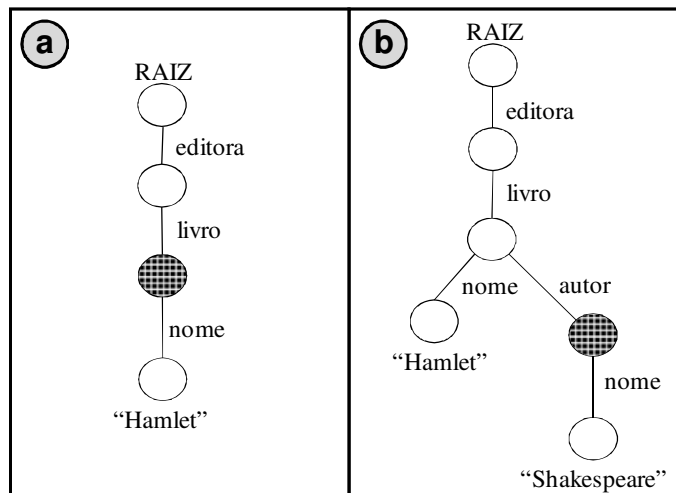


**Figura 66: Nós alvo de consultas XPath.**

O XPath provê funcionalidade que possibilita a utilização de diferentes *eixos*. Utilizando um *eixo* do XPath podem ser definidas relações de restrição entre um nó da expressão de consulta com outro nó. Nas expressões anteriormente citadas, utilizamos o *eixo pai-filho* ( / ) para definir relações em que um nó seja filho direto de outro nó. Além deste, o *eixo atributo* ( @ ) permite definir relações em que um elemento tenha um determinado nó de atributo. Em adição ao eixo pai-filho ( / ) e o eixo atributo ( @ ), XPath define outros 10 tipos de eixos, incluindo, por exemplo, o *ancestral-descendente* ( // ) e os eixos *irmão-anterior* e *irmão-sucessor* [Gou07].

Na Figura 67, são ilustrados os diferentes conjuntos de nós selecionados sob a restrição de seleção imposta por cada um dos *eixos* do XPath anteriormente citados (cada

conjunto é destacado através de uma borda em pontilhado) em relação a um determinado nó alvo (destacado com preenchimento pontilhado).

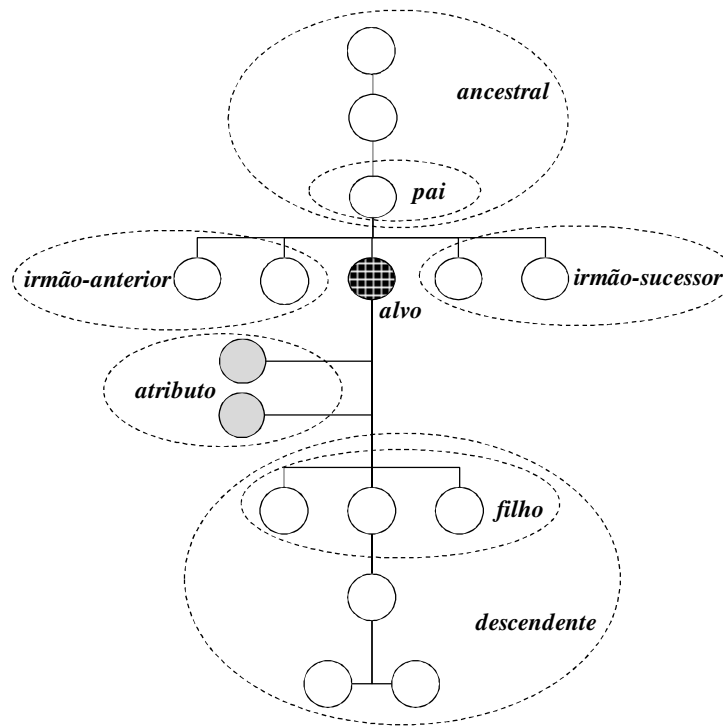


**Figura 67: Eixos do XPath**