IMPLEMENTAÇÃO DE UM ÍNDICE INVERTIDO PARA INDEXAR DOCUMENTOS DE TEXTO LONGO UTILIZANDO ÁRVORE B+ E FUNÇÃO HASH.

Edmilson dos Santos de Jesus - ed@edmilson.eti.br
Jorge Sampaio Farias - jfarias@actha.com.br
Universidade do Estado da Bahia - UNEB, Departamento de Ciências Exatas e da Terra
Rua Silveira Martins nº 2555 – Cabula – Campus I
CEP: 41195-001 – Salvador – Ba.

Resumo: Atualmente muitos estudos vêem sendo desenvolvidos na área de indexação de documentos de texto, porém a maioria das estruturas propostas requerem sofisticados recursos de hardware para seu processamento. Esse artigo apresenta a construção de um índice para documentos de texto longo com baixo custo de processamento. O índice proposto é implementado utilizando-se a técnica de índice invertido sobre árvores B+ e função hash. Ele permite atualização incremental onde os documentos podem ser adicionados sem a necessidade de reorganização do índice. Além disso ele usa exclusão lógica para evitar o bloqueio do índice, permitindo que o mesmo seja pesquisado enquanto está sendo atualizado.

Palavras-chave : *índice invertido, árvore B+, função hash, compressão de bits, busca por frases.*

1 INTRODUÇÃO

Segundo Harman et. al. (1992) arquivos invertidos são tradicionalmente utilizados para a implementação de índices lexicográficos. Para permitir a busca por frases o índice invertido contém, para cada palavra, um apontador para cada um dos documentos nos quais a palavra ocorre, juntamente com as posições da palavra nesse documento. Nesse tipo de índice a busca torna-se bem mais eficiente, mas ela traz consigo um custo adicional, a dimensão do próprio índice. Ainda segundo Harman et. al. (1992) essa dimensão é da ordem de 10 a 100% do tamanho do texto original. Além disso, o índice precisa ser atualizado a cada modificação dos documentos originais.

No pressente artigo é apresentada a construção de um índice invertido, o qual permite atualização incremental e exclusão lógica de documentos. Tal índice é implementado utilizando-se árvores B+ e função hash. Inicialmente na seção 2 é mostrado uma breve histórico de trabalhos desenvolvidos nessa área. Na seção 3 é descrito o modelo do índice proposto, sua arquitetura e implementação. Os resultados experimentais obtidos na busca e

indexação de um conjunto de documento é mostrado na seção 4. E finalmente na seção 5 é apresentada a conclusão com a análise dos resultados obtidos, limitações e sugestões para futuros trabalhos.

2 TRABALHOS ANTERIORES

Existe uma grande quantidade de trabalhos que tratam sobre a idexação de documentos. Em Brin and Page (1998) é descrita a arquitetura do Google, um mecanismo de busca que indexa bilhões de documentos da internet.

O Google mantém na memória principal uma tabela de vocabulários. Uma função converte a palavra procurada em uma entrada válida na tabela, na qual a respectiva saída aponta para uma estrutura de índice invertido contendo a lista dos documentos onde a palavra aparece. Esses documentos são identificados univocamente e estão seguidos das posições onde a palavra ocorre dentro dos mesmos. Ao final da busca o algoritmo *PageRank* é utilizado para classificar o resultado. O Google também utiliza vários clusters, permitindo o balanceamento de carga e o processamento paralelo de algumas tarefas.

Em Nagarajarao et al. (2002), é apresentada uma estrutura de índice invertido que suporta atualização incremental na qual vários documentos podem ser adicionados ao índice sem a necessidade de reindexação. Além do que, o índice pode ser pesquisado enquanto está sendo atualizado.

3 CONSTRUÇÃO DO ÍNDICE

O índice proposto é composto por um vetor de 16 milhões de entradas, onde cada saída do vetor aponta para um índice invertido. Esse vetor é mantido na memória principal em um espaço de 64 MB.

3.1. ARQUITETURA

Cada termo indexado deve ter no máximo 4 caracteres de tamanho, as palavras maiores serão quebradas em partes para que sejam armazenadas no índice. Cada caracter é compactado em 6 bits, tendo-se assim 64 representações possíveis para cada caracter, entre letras e números. Assim, são necessários apenas 24 bits para representar cada termo o que dá 16 milhões de termos possíveis, que é exatamente o tamanho do vetor. Além disso, um inteiro de 4 bytes é suficiente para armazenar cada termo, e tal inteiro é utilizado como índice do vetor para o termo.

Cada entrada do vetor representa um termo e cada saída aponta para uma lista de documentos onde aquele termo ocorre, se a saída do vetor for nula quer dizer que aquele termo não está presente em nenhum documento do banco. A lista de documentos contém além do identificador, as posições e o número de vezes que o termo ocorre em cada documento. As informações sobre as ocorrências dos termos são armazenadas em um vetor de inteiros de 256 elementos, de 0 a 255, onde o primeiro elemento guarda o total de ocorrências do termo no documento.

O índice invertido é implementado utilizando-se árvore B+, para tornar a busca mais eficiente.

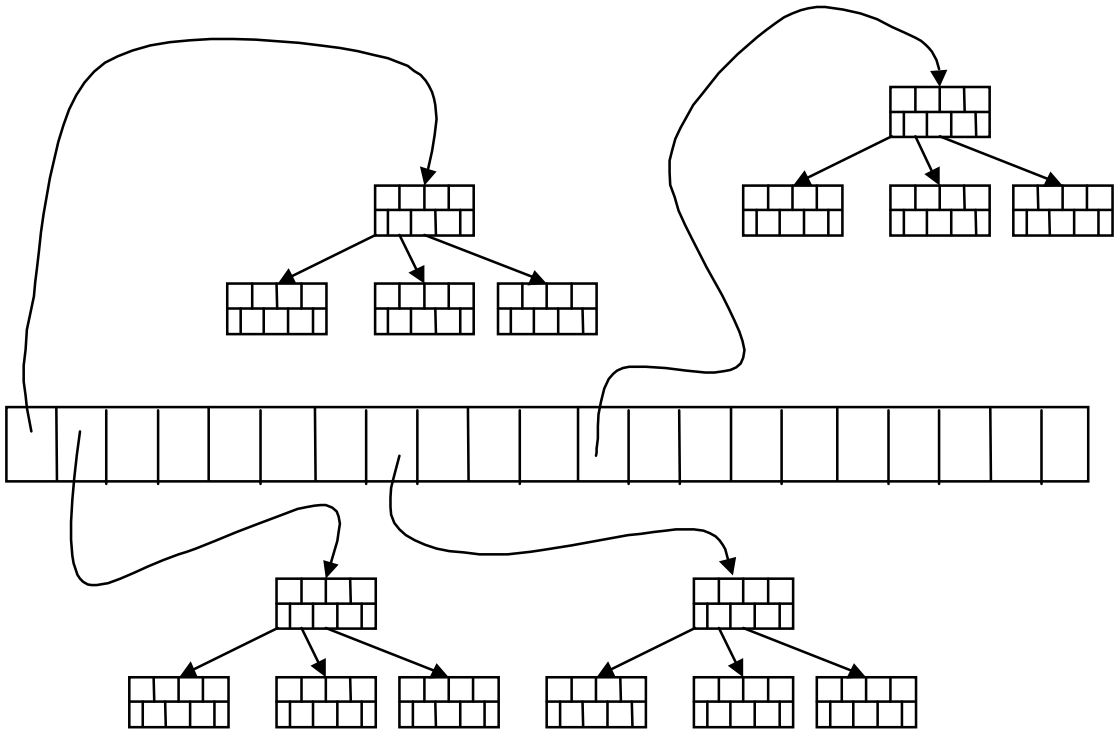

Figura 1 - Índice proposto

Sendo *x* um termo de no máximo 4 caracteres, *f(x)* a função que retorna o inteiro representado por *x*, e *v* o vetor que tem como saída ponteiros para listas de documentos. A lista l_i de documentos onde o termo x_i ocorre é dada pela função $l_i = v[f(x_i)]$. Se l_i for vazia não existem documentos onde x_i ocorre, caso contrário os documentos estarão na lista l_i.

No índices invertido proposto a pesquisa por frase é feita da seguinte forma:

(1) O primeiro termo é pesquisado, como resultado tem-se uma lista temporária de documentos e posições contendo o termo pesquisado.

(2) Utiliza-se a lista temporária para pesquisar o próximo termo, sendo retirados dela todos os documentos cujo o termo pesquisado não ocorre na posição adequada, ou seja, na posição subsequente ao termo anteriormente pesquisado.

(3) Repete-se a pesquisa para os próximos termos até que o último tenha sido pesquisado, ou até que a lista esteja vazia, indicando que a frase não foi encontrada.

Quanto menor for o número de documentos em que os primeiros termos pesquisados ocorrem, menores serão as listas temporárias e mais rápidas serão as pesquisas dos próximos termos. Dessa forma é desejável que os termos menos comuns sejam pesquisados inicialmente.

Supondo que se deseje pesquisar pela palavra “ajuda”, sendo esta maior que quatro caracter divide-se a mesma em dois termos “ajud” e “a”, após essa divisão converte-se os termos em números inteiros, obtêm-se as listas de documentos contendo os termos procurados usando-se os inteiros como índice no vetor, faz-se então a interseção das listas, observando-se as posições dos termos procurados. Nesse caso o temo “a” deve ser encontrado quatro posições subsequentes ao termo “ajud” no mesmo documento.

A exclusão de um documento no índice pode ser feita a qualquer momento, sem que seja necessário suspender as buscas no momento da exclusão. Isso é possível através da exclusão lógica do documento, que é feita, simplesmente, zerando-se as ocorrências de palavras para o documento no índice.

O identificador dos documentos indexados é mantido em uma árvore B+ e cada folha tem uma lista de ponteiros para as ocorrências do documento no índice. Assim quando um documento é excluído ou alterado, essa lista é utilizada para atualizar as ocorrências do documento no índice sem que o mesmo seja acessado diretamente. A desvantagem da exclusão lógica é que o espaço marcado como excluído não será mais utilizado, criando-se espaços vazios na árvore, até que ocorra uma reindexação e as árvores sejam recriadas. A Figura 2 apresenta uma ilustração da estrutura auxiliar utilizada na exclusão lógica.

Existe ainda uma lista de palavras comuns que não precisam ser indexadas porque aparecem na maioria dos documentos e por isso só ocuparia mais espaço no índice. Essa lista de palavras é mantida em um arquivo separadamente e no momento da indexação ela é utilizada para evitar que as palavras nela presentes e contidas no documento sejam indexadas.

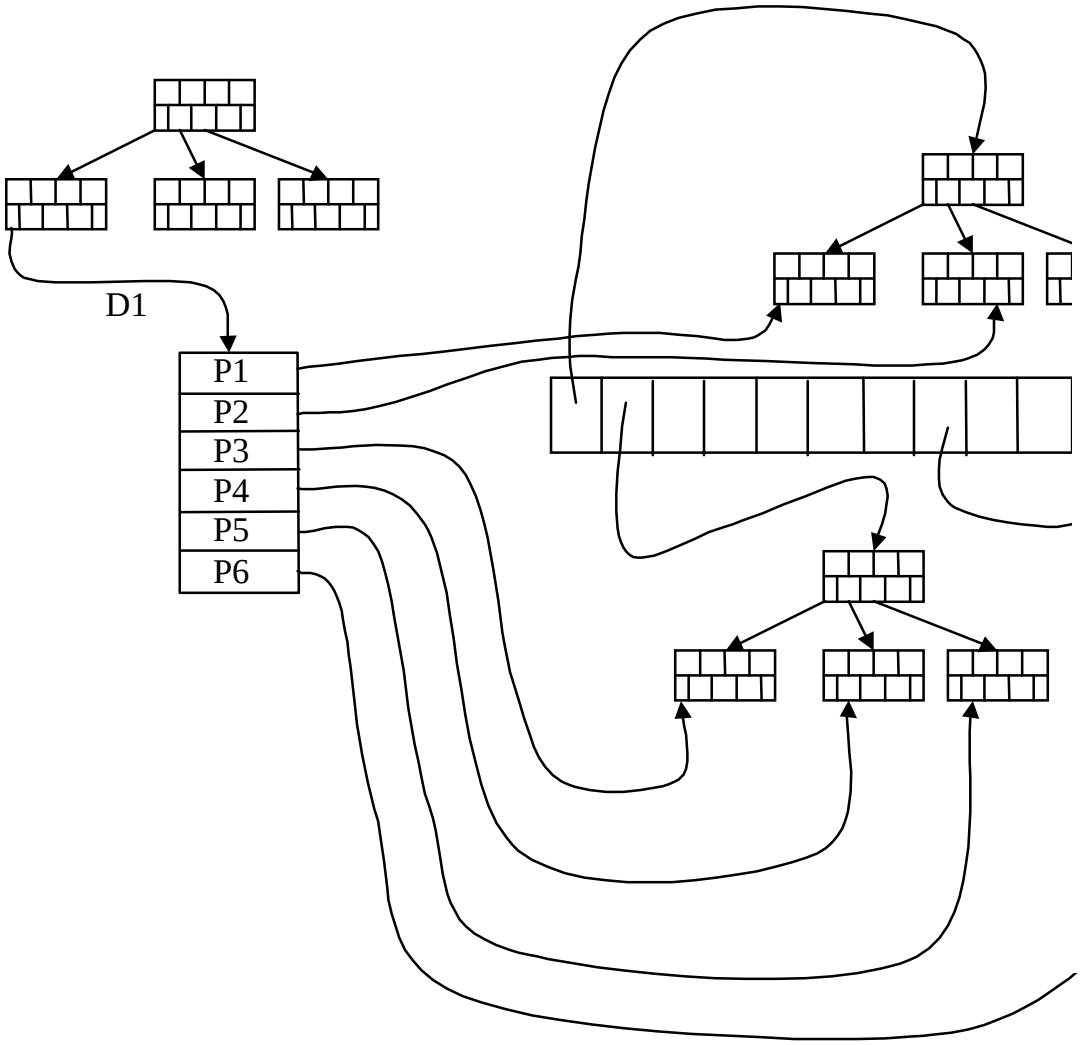

Figura 2 - Estrutura auxiliar utilizada para atualizar o índice.

3.2. IMPLEMENTAÇÃO

O índice proposto foi implementado utilizando-se a linguagem C, para compilação utilizou-se o GNU C Compiler (GCC), compilador de código fonte aberto e gratuito. O computador utilizado foi um PC com processador AMD Duron 700 MHz, 128 MB de memória RAM e 4GB de espaço em disco rígido.

O vetor é implementado através da declaração de um ponteiro para uma região de memória onde são reservados contiguamente 64MB de espaço. Isso é necessário porque são 16 milhões de entradas multiplicado por um inteiro de quatro bytes. Cada entrada, que representa cada termo, armazenará o endereço para o nó raiz de uma árvores B+.

A função de espalhamento converte um termo de quatro caracteres em um inteiro de 24 bits que é armazenado em quatro bytes. Tal inteiro é utilizado como índice do vetor, e o acesso é feito simplesmente utilizando-se aritmética de ponteiros, que é um recurso presente no na linguagem C.

Para cada termo existe uma árvore B+ contendo a lista dos documentos onde o mesmo ocorre. O endereço dessa árvore é mantida no vetor na posição correspondente ao índice do termo. Tal índice é calculado aplicando-se a função hash ao termo em questão. Essa árvore comporta 9 chaves por nó, e para cada chave existe um vetor de 256 posições que armazena as posições onde o termo ocorre em cada documento. Cada nó tem um tamanho aproximando de 10 Kbytes e tem capacidade de armazenar até 2295 ocorrências de termos.

A estrutura auxiliar é mantida em uma árvore B+ contendo as identificadores de todos os documentos indexados, e para cada identificador uma lista de ponteiros para todas as ocorrências do documento no índice. Assim quando um documento precisa ser atualizado essa árvore é percorrida.

4 RESULTADOS

O Gráfico 1 mostra experimentalmente o tempo consumido na indexação de alguns arquivos.

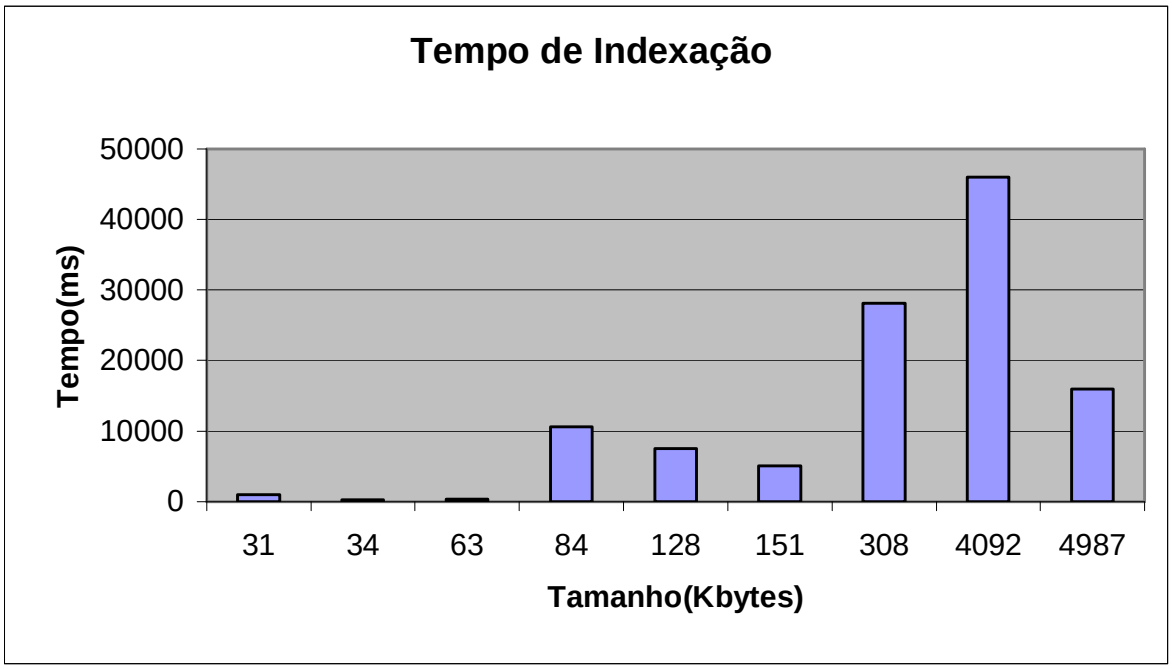

Gráfico 1 – Tempo de Indexação de documentos.

No Gráfico 1 os arquivos aparecem na mesma ordem em que foram indexados, da esquerda para a direita. Observando o gráfico é possível notar que o arquivo de 34 KB levou menos tempo para ser indexado que o arquivo de 31 KB, o mesmo ocorreu com o arquivo de 4987 KB. Isso ocorre porque quando uma palavra é indexada pela primeira vez, uma árvore é criada para armazenar as ocorrências da mesma nos documentos, e isso leva tempo. Mas nas próximas vezes que a mesma palavra for inserida no índice apenas alguns campos serão atualizados, apenas quando o nó a ser inserido estiver cheio é que ocorrerá uma divisão do mesmo. Entretanto vias de regra para registrar a ocorrência de uma palavra já presente no índice levará menos tempo que para indexar uma palavra não presente no índice.

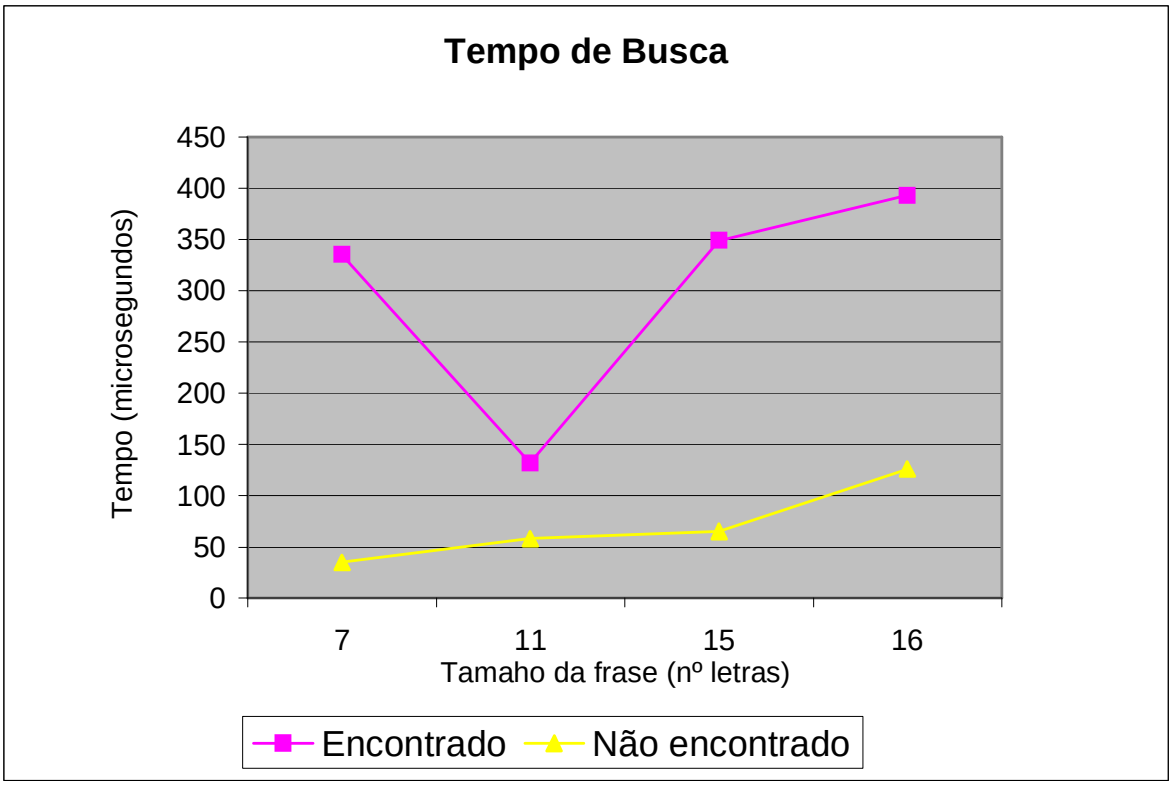

Gráfico 2 – Tempo de busca de documentos.

O Gráfico 2 mostra o tempo consumido na busca. Esse tempo corresponde às operações de interseção das listas de documentos onde os termos pesquisados ocorrem, observando-se as posições dos mesmos. Observa-se no gráfico que as frases não encontradas levam menos tempo para serem pesquisadas. Isso se deve, em parte, ao fato de que a maior parte do tempo consumido na busca corresponde ao tempo gasto com comparação de listas de documentos onde os termos procurados ocorrem, Como a lista de resultados tende a ficar vazia antes do final da busca, os tempos diminuem.

5 CONCLUSÃO

Os testes realizados mostraram que à medida que o índice cresce em diversidade de termos, mais rápido fica o processo de indexação de novos documentos. Por outro lado, conforme previsto, os testes também mostraram que o índice ocupou mais espaço do que o tamanho original dos arquivos indexados.

A utilização de um vetor de endereçamento direto, para localizar o índice invertido de cada termo, tornou o processo de busca mais eficiente. Visto que a etapa de pesquisa no índice pelo termo procurado é eliminada, o que ocorre na prática é apenas a interseção das listas de documentos que contém os termos procurados.

Quanto aos resultados obtidos nos testes pode-se concluir que entre as variáveis que podem influenciar o tempo de indexação de um arquivo, destacam-se o tamanho das palavras e a quantidade de termos repetidos no mesmo. Em relação ao tempo de busca, esse está diretamente associado ao tamanho das listas a serem comparadas, assim sugere-se que os primeiros termos pesquisados sejam os mais raros possíveis, tornando as listas menores e reduzindo o tempo gasto com comparações.

A linguagem C, utilizada na implementação do índice, mostrou-se bastante adequada, pois dispunha dos recursos imprescindíveis à implementação, como aritmética de ponteiros, operações binárias e acesso a arquivos.

5.1. FUTURAS DIREÇÕES

A implementação do índice não levou em consideração aspectos de performance e compressão de dados, por estarem além dos objetivos desse trabalho. Porém, tais aspectos constituem temas relevantes para serem desenvolvidos em trabalhos posteriores.

A corrente implementação do índice não suporta pesquisa por partes de uma palavra, assim sugere-se o desenvolvimento desse tema em novos trabalhos. Tal busca poderia ser feita mantendo-se uma estrutura auxiliar com a lista de palavras indexadas. No momento da busca a lista seria consultada retornando o conjunto de palavras onde a substring aparece, nesse ponto as palavras seriam pesquisadas no índice.

A corrente implementação utiliza apenas a memória principal para armazenar o índice. Assim, quando a memória não é suficiente o SO faz paginação, o que reduz a performance do índice. Portanto, sugere-se como trabalho futuro a implementação do índice mantendo-se os nós da árvore B+ em disco, com exceção daqueles que estão sendo utilizados. Além disso, sugere-se:

a) O estudo sobre a possível utilização de alguma técnica de compactação aplicada às árvores B+, para otimizar o uso dos recursos.
b) Avaliação o comportamento do índice implementado, no que diz respeito a performance das consultas e atualizações no índice com grandes quantidades de dados.
c) Acrescentar ao índice a capacidade de indexar caracteres de pontuação.

6 REFERÊNCIAS

BAHLE, D.; WILLIAMS H. E.; ZOBEL, J. **Optimised Phrase Querying and Browsing of Large Text Database.** Proceedings of the Australasian Computer Science Conference, M. Oudshoorn (ed), Gold Coast, Australia, Janeiro de 2001.

BRIN, Sergey; PAGE, Lawrence **The Anatomy of a Large-scale Hypertextual Web Search Engine.** In Proceedings of the Seventh International World Wide Web Conference, 1998.

HARMAN, D. Et al. **Inverted Files, in Information Retrieval - Data Structures & Algorithms.** Prentice Hall, 1992.

HUGH E. W.; ZOBEL J.; ANDERSON P**. What's Next? Index Structures for Efficient phrases quieryng.** Proceedings of the Tenth Australasian Database Conference, Auckland, New Zeland, Janeiro 18-21, 1999.

JESUS, EDMILSON DOS S.; **Construção de um Índice para indexar arquivos de texto longo direcionado ao Firebird.** Universidade do Estado da Bahia - UNEB, 25 de Agosto de 2003.

NAGARAJARAO, A.; GANESH, J. R.; SAXENA, A. **An Inverted Index Implementation Supporting Efficient Querying and Incremental Indexing.** 6 de Maio de 2002.

PUTZ , Steve. **Using a Relational Database for an Inverted Text Index.** XEROX,1991.

TENENBAUM, A.M.; LANGSAM, Y.; AUGENSTEIN, M.J. **Estruturas de Dados Usando C.** São Paulo: Makron Books, 1995.