TRABALHO 2:
INDEXAÇÃO EM DISCO
ESCARECIMENTOS

RAUL H.C. LOPES

1. Os algoritmos de indexação

O objetivo fundamental do *Trabalho Prático 2* consiste em exercitar o processo de construção de índices residentes em disco para bases de dados, cujas chaves de indexação são strings de tamanho variável. Antes de mais nada é preciso atentar para os seguintes fatos:

- Essa situação ocorre freqüentemente no trabalho diário de bancos de dados.
- Em situações práticas os conjuntos de dados a indexar são normalmente muito maiores do que a memória RAM disponível e **não** é razoável assumir que é possivel manter o índice ou os arquivos de dados a indexar em memória.

O trabalho consiste fundamentalmente em construir dois programas diferentes para indexação de textos. Os dois programas construirão índices obrigatoriamente baseados em árvore-B. O primeiro programa, **Btree**, usará em sua implementação uma das variações de árvore-B, descrita na seção **6.2.4** de [2] ou em [1].

Em seu segundo programa, **BtreeX**, você tentará obter uma árvore-B que seja eficiente em termos uso de espaço em disco, redução de acessos a disco e tempo de pesquisa dentro de um nó.

Os problemas fundamentais do *TP2* são:

- As bases de dados a indexat são textos que não podem residir em memória.
- As chaves de indexação são palavras dos textos e como tal têm tamanho variável, embora se possa asumir um limite máximo para cada um em torno de algumas centenas de caráteres.

2. Índice baseado em árvore-B simples

A alternativa mais simples para a implementaçao do programa **Btree** usará consiste em usar uma variante obrigatoriamente um índice baseado em alguma variante de árvore-B, descrita descrita na seção **6.2.4** de [2] ou em [1]. É importante levar em conta que:

- Suas chaves são strings de tamanhos variáveis: pode haver palavras a indexar com um caracter e palavras com cem caracteres. Para superar esta dificuldade você pode, como descrito em [2]:
  - Armazenar nos nós referências para ocorrências de palavras, em vez das próprias palavras;
  - Fixar o espaço disponível por nó para as chaves, mas permitir que cada uma delas ocupe espaço variável. Neste caso, o split do nó ocorre quando não existe mais espaço disponível para a nova chave.
- Seu índice e seus documentos residirão necessariamente em disco.

2.1. Índice baseado em árvore-B para strings.

O programa **BtreeX** usará obrigatoriamente um índice baseado em alguma variante de árvore-B, definida em um dos artigos:

- Referência básica sobre árvore-B e variações;
- A bíblia [2];
- Prefix B-tree;
- Evolução de estudo de caches.

A opção mais interessante aqui é a *simple prefix b-tree*, descrita nas seções um e dois de Prefix B-tree. A proposta ali apresentada consiste em ter:

- Uma árvore-B com chaves que podem ser prefixos das chaves reais dos textos indexados: essas chaves são *separadores ou particionadores* do espaço de pesquisa.
- Um arquivo de índice com as chaves reais.

Note que no extremo esta alternativa pode ser implementada através de dois arquivos, que formam o índice, e os textos a indexar. Os dois arqivos de índice constituiriam o que o artigo Prefix árvore-B chama de representação de *simple prefix B-tree* em um B^**-tree*.

Note que a árvore-B tem como único objetivo particionar o espaço de pesquisa. O arquivo com as chaves reais indicaria que chaves estão sendo indexadas e todas as suas ocorrêcias no texto.

Por exemplo, considere que existe um único texto e que ele contém as palavras: *on,part,problem,solution,prototype.* Assuma inserção na ordem de apresentção. Após a inserção das duas primeiras o índice

ficaria com a estrutura:

$$([on]o[part])$$

A intenção da representação acima é indicar que na árvore-B existe apenas a raiz com a palavra (de um caracter) "o". À esquerda está uma folha, contendo "on" e à direita uma folha contendo "part". As duas folhas poderiam estar armazenadas em um arquivo (ou sistema de arquivos) separado que além de indicar a chave indicaria todas as suas ocorrências em textos. Finalmente, note a palavra "o" que fica na raiz: ela é o menor prefixo de uma chave que separa que estão à esquerda e à direita.

Após a inserção da próxima palavra o índice fiacria:

$$([on]o[part : problem])$$

Note que o uso do caracter ":" como delimitador de chaves. Assuma agora que cada folha comporta no máximo 16 bytes. A inserção da próxima chave demanda um split produzindo:

$$([on]o[part]pa[problem : solution])$$

Decisões importantes a tomar:

- tamanho dos nós das folhas;
- tamanho dos nós internos da árvore-B: a raiz da árvore cima já contém três bytes ocupados com caracteres de separadores de chaves e mais bytes para delimitadores e referências para os nós.

Uma representação razoável para o nó interno da árvore-B seria

- um vetor de caracteres, que conteria as palavras que formam separadores delimitadas por caracters que não ocorrem no texto: nulo por exemplo.
- um vetor de inteiros com as referências para os nós filhos.

Assumindo que a raiz não comporta mais nenhum separador, a inserção próxima chave produz:

$$(([on]o[part])pa([problem]prob[prototype : solution]))$$

Referências

1. Thomas H.Cormen, Charles E.Leiserson, Ronald L.Rivest, and Clifford Stein, *Introduction to algorithms*, MIT Press, 2001.
2. Donald E. Knuth, *The art of computer programmimg, volume 3: Sorting and searching*, Addison-Wesley, 1998.