TRABALHO 2:
INDEXAÇÃO EM DISCO

RAUL H.C. LOPES

1. Preliminares

Neste trabalho você estudará uma das mais importantes estruturas de indexação de arquivos: a árvore-B[1].

Neste trabalho, em particular, você deverá concentrar sua atenção na implementação de indexação em disco via árvore-B e em detalhes de desempenho envolvidos, como:

- redução de acesso a disco;
- eficiência de pesquisa dentro do nó;
- importância de uso inteligente de cache.

Algumas dicas importantes sobre a execução do trabalho seguem.

(1) Não deixe de ler este documento por completo antes de iniciar o trabalho.
(2) Siga estritamente as especificações deste documento: qualquer desvio delas pode significar a anulação de um exercício ou de todo o trabalho.

O trabalho pode ser executado em grupos, mas, como estabelecido na seção 5, grupos maiores recebem menos créditos. Um elemento importante da avaliação do trabalho trata da detecção de mutualismo parasitário: situação em que um aluno atua de vampiro parasita sugando o resultado do trabalho do grupo. Fato: plágio em qualquer item de um trabalho implica na anulação completa do trabalho do(s) grupo(s) envolvidos. Qualquer elemento de um grupo pode ser, durante a execução do trabalho ou após a entrega do mesmo, avaliado em relação ao conhecimento do que foi feito. Essa avaliação poderá ser feita de forma oral ou por escrito.

2. Os exercícios

O objetivo dos exercícios consiste em estudar o desempenho de duas estruturas para indexar uma série de documentos.

[1]Veja [2], [1], como referências gerais sobre árvore-Be o *survey* que aborda árvore-B e variações.

Exercício 1. *Implemente um programa que use um índice baseado em árvore-B*[2] *para realizar operações sobre as ocorrências de palavras de uma série de textos como descrito na seção 3. Seu índice terá obrigatoriamente a estrutura descrita na 4.1.*

O programa compilado resultante será denominado ***Btree****.*

Exercício 2. *Implemente um programa que use um índice baseado em árvore-B*[3] *para realizar operações sobre as ocorrências de palavras de uma série de textos como descrito na seção 3. Seu índice terá obrigatoriamente a estrutura descrita na 4.2.*

O programa compilado resultante será denominado ***BtreeX****.*

Exercício 3. *Escreva artigo em LaTeX, contendo estudo de desempenho de seus indexadores com diversos conjuntos de documentos. Seu artigo abordará ao menos:*

- *número de acessos a disco;*
- *uso de cache;*
- *tempo de execução.*

Procure caracterizar esses três itens em relação a diferentes tipos de seqüências de operações.

3. As operações a implementar

Seus dois programas[4] admitirão o mesmo tipo de **input** e implementarão as mesmas operações, que consistem em inserir palavras de tamanho variável em um índice, excluir palavras e listar ocorrências de palavras desse mesmo índice.

3.1. **Dados de entrada saída de cada programa.** Cada programa receberá como único argumento da linha de comando o nome de um arquivo com o conjunto das operações a realizar. Esse arquivo terá a seguinte estrutura:

- A primeira linha: nome do diretório contendo os documentos de entrada.
- A segunda linha: nome do diretório contendo os arquivos de índice gerados.
- A terceira linha: o nome do diretório que conterá os resultados das operações do tipo **l**, definido a seguir.

  Este diretório conterá um arquivo para cada operação **l**, veja abaixo, especificada.

[2]Veja [2, 1].

[3]Sobre árvore-B em geral veja [2] e veja o *survey* sobre variações.

[4] Esses programas são: **Btree**, parte do exercício 1 e que usa índice descrito na seção 4.1; **BtreeX**, parte do exercício 2, cujo índice é descrito em 4.2.

```
indir
outdir
listdir
i text
i ztext
l list
d ztext
l alist
```

FIGURA 1. Exemplo de entrada

- As linhas seguintes, em número variável e em qualquer ordem terão sempre o formato: uma letra, definindo uma operação, seguida de um espaço, seguida de um nome de arquivo, que estará contido no dirtetório de entrada. As operações (e letras) associadas são:
  - **i**: insira no índice todas as palavras do documento;
  - **d**: exclua do índice todas as ocorrências de palavras do documento;
  - **l**: liste em ordem crescente todas as ocorrências de palavras que tenham como prefixo as palavras contidas no documento. Uma ocorrência de uma palavra é um par composto da palavra e o nome do documento em que ela ocorre, separados por um espaço em branco.

  Por exemplo, um arquivo com o formato da figura 1 define:

  (1) O diretório de nome **indir** contém os arquivos de entrada do seu programa.
  (2) O diretório de nome **outdir** conterá os arquivos gerados pelo seu programa.
  (3) A primeira operação consiste em inserir, no índice situado no diretório **outdir**, todos as palavras do arquivo **text**, localizado no diretório **indir**.
  (4) A segunda operação consiste em inserir, todos as palavras do arquivo **ztext**, localizado no diretório **indir**.
  (5) A terceira operação consiste em listar em ordem crescente todas as ocorrências de palavras que tenham como prefixo palavras do arquivo de nome **list**, contido no diretório de nome **indir**.
  (6) A quarta operação consiste em excluir, do índice situado no diretório **outdir**, todas as palavras do arquivo **ztext**, localizado no diretório **indir**.

(7) A quinta operação consiste em listar em ordem crescente todas as ocorrências de palavras que tenham como prefixo palavras do arquivo de nome **alist**, contido no diretório de nome **indir**.
(8) Ao final, o diretório de nome **listdir** conterá dois arquivos:
   – **list**, conterá o resultado da primeira operação **l**, realizada com palavras do arquivo **list**.
   – **zlist**, cont erá o resultado da segunda operação **l**, realizada com palavras do arquivo **alist**.

   Por exemplo, o arquivo **list**, em **listdir**, poderia conter a linha

brrrp ztext

   que *diz* que a palavra **brrrp** ocorre no arquivo **ztext** .

O diretório de nome **outdir** conterá ao final um ou mais arquivos com o índice gerado pelo seu programa. Note que o índice gerado deve poder ser reutilizado para operações posteriores: deve ser possível executar seu programa várias vezes sobre o mesmo índice.

3.2. **Dados de saída de seus programas.** Todo *output* gerado pelo seu programa será gravado em arquivos como descrito na seção 3.1.

4. Os algoritmos de indexação

Os dois programas construirão índices obrigatoriamente baseados em árvore-B. O primeiro programa, **Btree**, do exercício 1, 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**, do exercício 2, 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.

4.1. **Índice baseado em árvore-B.** O programa **Btree** usará 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.

Critério	Créditos
Exercício 1	30
Exercício 2	30
Exercício 3	20
Seção 5.2	10
Seção 5.3	10

TABELA 1. Tabela de atribuição de créditos

- Seu índice e seus documentos residirão necessariamente em disco.

4.2. **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 árvore-B;
- Evolução de estudo de caches.

Uma restrição fundamental é de que ele deve ser necessariamente mais eficiente do que o programa **Btree**. Você deve tentar encontar uma forma de tratar eficientemente os seguintes fatos:

- Suas chaves são strings de tamanhos variáveis: pode haver palavras a indexar com um caracter e palavras com cem caracteres.
- Seu índice e seus documentos residirão necessariamente em disco.
- Suas pesquisas em memória devem tentar usar o melhor possível o cache de sua máquina.

5. CRÉDITOS

A atribuição de nota para o trabalho ocorrerá em duas fases:

(1) Atribuição de créditos pelo trabalho submetido, de acordo com a tabela 1.
(2) Definição do multiplicador de acordo com a tabela 2.

5.1. **Definição do número de créditos.** Cada trabalho submetido será avaliado para receber de zero a cem créditos de acordo com a tabela 1.

5.2. **Desempenho do programa Btree.** Este item atribuirá pontos de acordo com o desempenho do programa **Btree**, de acordo com o critério de desempenho: o programa mais rápido ganha 10 créditos.

Fator	Multiplicador
Grupos de 1 aluno	11/100
Grupos de 2 alunos	10/100
Grupos de 3 alunos	75/100
Grupos de 4 alunos	60/100
Grupos de mais 4 alunos	0
Plágio	0

TABELA 2. Tabela de multiplicadores

5.3. **Desempenho do programa BtreeX.** Este item atribuirá pontos de acordo com o desempenho do programa **BtreeX**, de acordo com o critério de desempenho: o programa mais rápido ganha 10 créditos.

5.4. **Definição de multiplicador.** O nota de cada trabalho é dada pelo número de créditos obtidos pelo trabalho multiplicado pelo fator da tabela 2.

Em caso de plágio todos os trabalhos de grupos envolvidos são anulados. Poderá ser enquadrado como plágio qualquer trabalho em que um ou mais alunos do grupo não tenham conhecimento de qualquer item do trabalho. Esse conhecimento deverá ser demonstrado em prova e/ou enttrevista.

6. A submissão para correção

A data final de entrega deste trabalho é 04/04/05.

A mensagem de submissão do seu trabalho deverá conter:

- **Subject: tbo.tp2**
- *Attachment*: **tp2.tar.bz2**

O corpo da mensagem será desprezado. O arquivo anexdo deverá ser obrigatoriamente denominado

tp2.tar.bz2

Ele conterá todo os fontes necessários para compilar seu trabalho prático, incluindo fontes em LaTeX, C/C++, Makefile e arquivo de identificação.

As seguintes regras devem ser rigorosamente seguidas, não cabendo recurso no caso de alguma delas não ser respeitada:

- Nenhum compilado deve ser enviado.
- Não use os pacotes **listings** e **noweb** no artigo deste trabalho.
- O *tarball* do seu trabalho conterá a seguinte estrutura:
  - arquivo de **id**

Conterá uma linha inicial com *e-mail* de contato do grupo e, depois, uma linha de identificação para elemento do grupo, contendo número de matrícula e nome completo separados por '**:**' (dois pontos.) Por exemplo:

```
jolero@gmail.com
99900089:Ze' do lero
```

  - arquivo **Makefile**
  - diretório **Btree**
    Conterá a implementação do exercício Exercício 1.
  - diretório **BtreeX**
    Conterá a implementação do exercício Exercício 2
  - diretório **doc**
    Conterá os fontes de toda a documentação do trabalho.

  Adicionalmente seu *tarball* poderá conter outros diretórios relevantes para a sua implementação.
- seu *tarball* será aberto e compilado com as seguintes linhas:

```
tar -jxf tp2.tar.gz
make all
```

  Essas linhas gerarão os seguintes arquivos:
  - **Btree**
    Executável que implementa a solução do exercício Exercício 1.
  - **BtreeX**
    Executável que implementa a solução do Exercício 2.
  - **artigo.pdf**
    Artigo em formato **PDF**, que avalia qualidade de *tour* e desempenho dos algoritmos.

7. Trabalho extra

Este trabalho não é obrigatório. Ele poderá contribuir para a sua nota nas seguintes condições:

- Se você concluir o semestre com nota igual ou superior a seis, estará aprovado e a nota deste trabalho poderá ser usada para elevar sua nota final, como descrito no documento de regras de avaliação da disciplina.
- Alunos que concluírem o semestre com nota entre cinco e seis, serão aprovados com nota seis desde que: concluam 60% das listas de exerçício e alcancem nota sete neste trabalho.
- Se você fizer prova final, ela será composta de duas partes: este trabalho, valendo quarenta por cento da nota da prova, e uma

prova escrita a ser realizada no dia 16/03/05, às 13 horas, na sala de aulas usada durante o semestre.

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.