Universidade de São Paulo
Instituto de Ciências Matemáticas e de Computação
Departamento de Ciências de Computação
SCC-203 – Algoritmos e Estruturas de Dados II / 2011
Prof.ª Rosane Minghim

Trabalho 3

Descrição

1. Retomando o enunciado do Trabalho 2, um pesquisador deseja manter um cadastro de suas referências bibliográficas (artigos e livros) em disco, e acabou de contratá-lo para resolver o problema. Para cada referência, o pesquisador deseja manter os seguintes atributos:
   - Código da referência (composição das três primeiras letras do nome do primeiro autor e ano da publicação, por ex. SHI90 – esse campo é chave).
   - Título (string com o título).
   - Autor (sobrenome e iniciais do primeiro autor, por ex.: Schimman,D.E.).
   - Ano de publicação (por ex.: 1990).
   - Veículo (string longa com o nome do congresso ou periódico, e outros dados adicionais, como páginas, volume e número, local, etc.).
2. Você deverá desenvolver um sistema que permita ao pesquisador:
   - Inserir uma nova referência, atualizando um índice.
   - Remover uma referência a partir da chave. O registro deverá ser localizado acessando o índice. A remoção deve colocar o caractere # (sustenido) na primeira posição do registro removido. O espaço do registro removido não deverá ser reutilizado para novas inserções, as quais deverão ser colocadas sempre no fim do arquivo (supõe-se, assim, que um processo de compactação eliminaria os espaços disponíveis em outro processo, o qual não é necessário implementar).
   - Buscar uma referência a partir da chave. A busca deve ser realizada acessando o índice.
3. Ao final da execução, dois arquivos deverão ter sido criados: um arquivo de índice e um arquivo de dados.

➢ Estrutura do índice

O arquivo de índice deve ser binário e armazenado em uma **árvore-B**. Note que os "ponteiros" para as páginas devem ser os *offsets* da sua localização no arquivo. A árvore-B deve seguir as seguintes especificações:

- Cada página da árvore deve comportar 4 entradas do índice (em uma situação real, as páginas comportariam muito mais entradas, então implemente sua árvore-B de modo que mudar esse valor seja uma questão de mudar uma constante).
- A chave deve ser declarada como um vetor de 6 caracteres (um a mais para o '\0'). No entanto, apenas os 5 caracteres da chave devem ser gravados no disco, ocupando, assim, 5 bytes.
- Os endereços (*offsets*) devem ser do tipo *long* de 4 bytes (tanto os *offsets* das próximas páginas da árvore, quanto os *offsets* dos registros). *Offsets* nulos devem ter valor "-1".
- As páginas da árvore devem começar indicando o número de entradas do índice armazenadas nela. Esse valor deve ser do tipo *short* de 2 bytes.
- Os espaços vagos nas páginas devem ser preenchidos do seguinte modo: os *offsets* nulos devem ter valor "-1" (tanto *offsets* para páginas quanto *offsets* de registros vagos), e os espaços das chaves devem ser preenchidos com o caractere # (sustenido). Nas páginas-

folha, os *offsets* das próximas páginas (são todos nulos) também devem ter valor "-1". Lembre-se que todo *offset* é um *long* de 4 bytes e deve ser gravado como tal.

- Páginas excluídas devem ser marcadas com um caractere # (sustenido) na primeira posição, seguido do *offset* da próxima página livre. As páginas livres devem ser mantidas em uma espécie de pilha: sempre que uma página é excluída, ela passa a ser o topo da pilha, e o *offset* do antigo topo deve ser colocado após o caractere # (sustenido).
- Novas páginas devem ser colocadas na primeira página livre (topo da pilha de disponíveis). Nesse caso, antes de gravar a página, lembre-se de pegar o *offset* da próxima página livre, que passará a ser o topo da pilha de disponíveis. Caso não haja nenhuma página livre, novas páginas devem ser colocadas no final do arquivo.
- O cabeçalho do arquivo de índice deve conter o *offset* da página raiz, seguido do *offset* do topo da pilha de disponíveis. A árvore começa imediatamente em seguida, sem caracteres delimitadores nem quebras de linha.
- Caso a pilha de disponíveis esteja vazia, o *offset* deve ser "-1".
- Caso todos os registros tenham sido removidos (e, portanto, todas as páginas foram excluídas), o *offset* da página raiz é "-1". Nesse caso, todas as páginas (previamente criadas e então excluídas) vão estar marcadas com um # (sustenido) e a pilha de disponíveis conterá todas elas.

Como os campos do índice possuem tamanho fixo, tudo deve ser armazenado sem caracteres delimitadores nem quebras de linha, utilizando a estrutura tradicional de árvore-B, do seguinte modo:

$$N_E\ O_P\ «C_R\ O_R»\ O_P\ «C_R\ O_R»\ O_P\ «C_R\ O_R»\ O_P\ «C_R\ O_R»\ O_P$$

Onde N_E representa o número de entradas do índice armazenadas na página, cada O_P representa o *offset* para outra página da árvore, e cada par $«C_R\ O_R»$ representa o par composto pela chave de um registro e seu *offset*. Os símbolos *«* e *»* foram colocados apenas para facilitar a leitura, tudo deve ser armazenado sem delimitadores. A linha acima representa uma página da árvore-B, sendo que a próxima página deve começar imediatamente em seguida, sem caracteres delimitadores nem quebras de linha. Desse modo, cada página ocupa exatamente 58 bytes.

Em uma situação real, a árvore-B não caberia inteira em memória primária, e seria necessário implementar um sistema de paginação, isto é, somente um número determinado de páginas ficaria na memória primária por vez e, quando fosse necessário carregar uma que não estivesse, uma das páginas na memória deveria ser descarregada no disco para ceder lugar. No caso do trabalho, não é necessário implementar sistema de paginação, mas seu programa deve ser implementado de modo que inserir um sistema de paginação não exija mudar nenhuma das funções da árvore-B. Nesse sentido, algumas recomendações importantes:

1. Crie uma função que recebe o *offset* de uma página e retorna um ponteiro para a mesma. Pode-se manter um campo adicional na página (somente em memória primária) contendo seu *offset*. Havendo um sistema de paginação, essa função deveria verificar se a página já está na memória primária; se existe espaço disponível, caso não esteja; e efetuar a troca, se necessário.
2. De modo geral, as funções não devem receber como parâmetro páginas nem ponteiros de página. Ao invés disso, devem receber *offsets* de páginas. Então, utiliza-se a função do item anterior para pegar os ponteiros.
3. Para criar uma nova página, declare e aloque o espaço para a mesma, manipule o que for necessário, e então a grave no disco. Para gravar a nova página, crie uma função que

recebe o ponteiro para a página recém-criada e retorna o *offset* da mesma após a gravação no disco.

É importante notar que a estrutura de árvore-B existe apenas no disco (por meio dos *offsets*). Em memória primária, as páginas são armazenadas em uma lista ou vetor. Isto porque, com um sistema de paginação, somente uma pequena parte da árvore ficaria em memória primária, sendo impossível manter de fato sua estrutura. A função citada no primeiro item deve percorrer o vetor (ou lista) a procura da página com o *offset* especificado. **Obrigatoriamente o arquivo de índice deverá ser chamado de "index.dat".**

- **Estrutura do arquivo de dados**

O arquivo de dados deve ser ASCII, e organizado em registros de tamanho fixo de 256 bytes, com os campos título, autor e veículo de tamanho variável (título e veículo podem ser declarados com tamanho 100 e autor com tamanho 30, mas no arquivo só serão gravados até o '\0'), e os demais de tamanho fixo. Os campos no registro devem vir separados pelo caractere delimitador @ (arroba). Após o último caractere @, que indica o fim do registro, preencher o restante do espaço com o caractere # (sustenido). Por exemplo:

key01@título1@autor1@ano1@veículo1@##...##key02@título2@autor2@ano2@veículo2@##...

O arquivo de dados não deverá conter cabeçalho. **Obrigatoriamente o arquivo de dados deverá ser chamado de "data.dat".**

- **Comandos Permitidos**

Os comandos permitidos são representados por duas letras maiúsculas. Toda linha de entrada obrigatoriamente inicia com um comando. Só serão fornecidos como entrada comandos aqui especificados (não há necessidade de tratar comandos inválidos, pois só serão fornecidos comandos válidos). Os únicos comandos permitidos como entrada são apresentados a seguir. O símbolo ¶ denota um único espaço em branco. **Para facilitar, nenhum dos campos (chave, título, autor, ano e veículo) conterá espaços.**

1. **Insere referência**

IR ¶ <chave> ¶ <título> ¶ <autor> ¶ <ano> ¶ <veículo>

Insere uma nova referência contendo todos os campos, como especificado acima.

2. **Remove referência**

RR ¶ <chave>

Remove uma referência a partir da sua chave.

3. **Busca referência**

BR ¶ <chave>

Busca uma referência a partir da sua chave e imprime o registro da seguinte forma:

<chave> ¶ <título> ¶ <autor> ¶ <ano> ¶ <veículo>

*Coloque uma quebra de linha no fim.

4. Finaliza a execução

FM

Finaliza a execução do programa, atualizando o arquivo de índice no disco e fechando-o, juntamente com o arquivo de dados.

➢ **Entrega e Observações Importantes**

- Implemente o trabalho usando a linguagem C padrão ANSI ou em C++. A clareza do código será avaliada. Use comentários relevantes em seu código, pois serão avaliados.
- Os trabalhos deverão ser feitos individualmente, e devem estar devidamente identificados com nome e número USP comentados no início. Serão anulados aqueles trabalhos nos quais forem detectados quaisquer tipos de **cópia ou plágio**, não importa a origem.
- Os trabalhos deverão ser entregues pelo SQTPM em arquivo único (".c" ou ".cpp"). No entanto, o sistema não avaliará o programa. Os trabalho deverá ser apresentado pessoalmente ao assistente PAE da disciplina, em horário a ser marcado.
- Data máxima de entrega: **03/07**.

Tenha um bom trabalho!