CAPA

USP – Universidade de São Paulo
Professora Dra. Cristina D. A. Ciferri

Trabalho de SCC0215 – Organização de Arquivos (Parte 2)

Pedro Pastorello Fernandes – NUSP 10262502

Data de finalização do trabalho: 24/06/2018

Esta é a segunda parte do trabalho prático da disciplina de Organização de Arquivos. O projeto segue todas as especificações pedidas, rigorosamente.

Sobre a primeira parte do projeto:

A primeira parte do projeto foi entregue com uma divergência a respeito da especificacao do mesmo: ele armazenava o campo codEscola como uma string de texto no arquivo de dados, e não como um inteiro (binário de 4 bytes). Contudo, para a segunda parte do projeto, esse problema foi corrigido e, além disso, o código fonte foi melhorado e está mais limpo e mais modularizado. Portanto, esta parte do projeto prático foi baseada em uma versão mais limpa e funcional da primeira parte que segue exatamente o que é pedido em sua especificação.

O que foi feito:

A inserção no arquivo de indices Arvore B, corretamente acoplada às funcionalidades requisitas e a bufferização do arquivo de indices em RAM, acoplada à função de leitura de uma página do arquivo de índices.

O que não foi feito:

A remoção e a atualização de chaves no arquivo de índices.

Sobre a compilação e execução:

A compilação do programa é feita com o comando *make all*, no diretorio do projeto. A execução é feita como especificado, com o comando *./trabalho2 parametro1 parametro2 parametro3 parametroN*.

Sobre a implementação do Buffer-pool:

O buffer-pool de páginas da árvore-B está organizado como uma fila (FIFO), assim como especificado para o meu grupo. A fila comporta 4 páginas fuciona em um vetor estático que rotaciona em seus índices para manter seu funcionamento. Existe também uma posição especial, dedicada à página raiz da árvore.

O buffer também armazena os RRNs das páginas que estão armazenadas no mesmo, por meio de um vetor de 5 posições, sendo cada posição referente à mesma posição na fila (exceto pela quinta posição (i = 4), que armazena o RRN da página raiz, que não está no vetor da fila e sim em sua posição especial.

O buffer está acoplado na função de leitura de uma página da árvore. Tal função passa primeiro pelo buffer-pool e, caso não encontre a página a ser lida no buffer, a le do arquivo de índices e a armazena no buffer.

Não é necessário acoplar o buffer-pool à funcionalidade de escrita do programa, pois neste caso é sempre necessário fazer um acesso ao disco.

Sobre a funcionalidade de inserção:

A inserção é feita pela função *insereChaveIndice()*. Caso a arvore esteja vazia, a propria funcao cria a pagina raiz, insere a chave e inicializa a arvore. Senão, é chamada a função *insereChavePagina()* para a página raiz. Essa função inicia uma cadeia de chamadas recursivas dela mesma, até que seja encontrada a posição correta de inserção da chave.

Para a inserção de fato, são utilizadas as funçoes *inserePaginaNaoCheia()*, para o caso da inserção em um página não cheia, e *split()*, para o caso da inserção em uma página cheia e que deve sofrer um split. A função de split trata tudo o que é necessário para a realização do algoritmo e sobe recursivamente a árvore enquanto precisar realizar mais splits.

Sobre a funcionalidade de busca:

A funcionalidade de busca é simples e realizada e realizada pela função *buscaIndice()* sobre a página raiz da árvore. Essa função procura a chave busca na página atual e, caso não a encontre, retorna o RRN indicado pela chave. Caso não a encontre, encontra a ramificação correta a ser seguida e continua a busca, recursivamente, na página filha correta.

Sobre o uso de memória RAM do programa:

Apesar de reconhecer a importância do uso da função *free()* e do bom gerenciamento de memória principal por programas escritos em C, essa função não foi utilizada sobre os ponteiros criados durante o código pois não tive tempo disponível para debugar e otimizar efetivamente o uso de memória pelo código. A prioridade durante o desenvolvimento foi sempre a corretude dos algoritmos e a entrega do projeto.