INSTITUTO DE CIÊNCIAS
MATEMÁTICAS E DE COMPUTAÇÃO

Segunda Parte do Trabalho Prático (Parte 2)
SCC0215 - Organização de Arquivos

Bacharelado em Ciências de Computação - Turma B
Professora: Cristina Dutra de Aguiar Ciferri
Data de entrega: 24/06/2018

Nomes e Números USP:

Eduardo Manso Fernandes - Nº 8531675
Gabriel Francischini de Souza - Nº 9361052
Matheus Carvalho Raimundo - Nº 10369014
Paulo Renato Campos Barbosa - Nº 9779475

São Carlos/SP

ÍNDICE

Adendo:

As funcionalidades do trabalho e suas implementações estão contidas no arquivo *funcionalidades.c*, e a árvore-B em *arvore-b.c*. Na *main.c*, foi usado o 'argv' e um switch para determinar qual operação será realizada. Neste switch, chama-se as funções específicas para cada funcionalidade.

O programa foi desenvolvido, compilado e testado no sistema operacional Linux Mint, utilizando o compilador GCC 5.4.0. Ele pode ser compilado utilizando-se o comando make ou make all, e pode ser executado utilizando tanto make run, como também ./programaTrab2.

Foram feitos testes em todas as funcionalidades e todas estão funcionando corretamente e seguindo estritamente o que está descrito na especificação do trabalho.

O nome do arquivo binário deste trabalho foi escolhido como "registros.dat", o da árvore-b é "indice-arvoreb.dat" e as informações do buffer-pool estão em "buffer-info.text". Tais arquivos são gerados e armazenados no mesmo diretório de execução do programa.

Decisões de projeto:

A especificação do trabalho não descreve nada relacionado a erros por parte do usuário, ou seja, caso ele digite uma operação que não exista, ou tente inserir um novo registro no arquivo binário sem antes ter criado ele, por exemplo. Apesar de que foi tratado alguns erros por parte do usuário, não foi especificado como estes erros deveriam ser tratados (que mensagem seria exibida, etc...), e exatamente por isso considerou-se que o usuário não erraria (até porque a correção será automática, e não manualmente).

Os nomes das funções no código são autoexplicativos, ou seja, quando queremos imprimir um registro, chamamos a função imprimirRegistro; quando queremos inserir um registro, chamamos inserirRegistro; assim como iniciarBufferPool inicializa o Buffer-Pool para uso, e assim por adiante. Deste modo não é necessário repetir todas as vezes o nome destas funções, afim de trazer mais clareza ao leitor.

Por fim, foi optado pela modularização do código, separando as funções que lidam diretamente com o arquivo binário, colocando-as em *funcionalidades.c*, das funções que lidam com o usuário na *main.c*, as funções da árvore-B em *arvore-b.c*, e os cabeçalhos em *cabecalho.c*.

SEÇÃO 1:

Este é o grupo 4 da turma B, portanto o algoritmo de gerenciamento do Buffer-Pool implementado foi o SCA (Second-Chance Algorithm).

As funções utilizadas para efetuar este algoritmo na árvore-b que valem a pena ser mencionadas são iniciarBufferPool, finalizarBufferPool, FIFOShift, lerPagina, escreverPagina, inserirChaveRecursao, removerChaveRecursao e buscaRRN. Vale lembrar que a raiz sempre está no Buffer.

Em buscaRRN retorna-se o RRN de acordo com a chave (codINEP) passada como parâmetro. Enquanto existirem filhos (altura maior que zero), lê-se (com lerPagina) os nós, percorrendo cada chave em cada nó e, quando encontrada a chave desejada, retorna-se o RRN do registro no arquivo de dados. Caso seja necessário descer mais, deve-se decrementar a altura e, caso não encontrada a chave, retorna-se -1.

Em inserirChave, insere-se a chave de busca e seu RRN na árvore-B. Caso não haja raiz criada ainda, deve-se criar uma. Caso necessita-se inserir a chave e fazer a promoção, a função inserirChaveRecursao é chamada. Vale lembrar que nestes 2 casos, utiliza-se a função escreverPagina, que contém o algoritmo SCA implementado. Nesta função, salva-se uma página na árvore-B. Se a página já estiver no buffer pool, não é necessário acessar o disco, mas apenas marcar como modificada. Caso contrário, esta página entra no Buffer-Pool e substitui outra, se necessário, de acordo com as políticas do SCA (REFERÊNCIA).

Em removerChave, remove-se a chave de busca (codINEP) da árvore-B e seu RRN é retornado. Percorre-se cada nó da árvore-B até encontrar o nó da chave buscada. A partir daí, remove-se (com removerChaveRecursao) e faz-se as concatenações de nós (caso necessário), checando também se necessita-se alterar a altura da árvore ao fim do algoritmo.

Em iniciarBufferPool, apenas é limpado o Buffer-Pool e a raiz, de forma que os espaços estejam livres para uso nos algoritmos. Também é atribuído o valor zero a Page Fault e Page Hit.

Em finalizarBufferPool, escreve-se as modificações feitas no Buffer em disco e acrescenta em “buffer-info.text” os números de Page Fault e Page Hit. OBS: Também escreve o nó raiz caso haja alguma modificação neste.

SEÇÃO 2:

A função transferirCSV é chamada para a funcionalidade 10 e tem como objetivo passar os registros do CSV para o arquivo DAT. Esta abre o arquivo de entrada (CSV), cria o arquivo DAT e cria o arquivo de índices. A partir daí, chama-se as funções que escrevem no cabeçalho e no cabeçalho do índice.

Chama-se também iniciarBufferPool, para liberar espaços no Buffer-Pool para os algorítmos.

Por fim, cada registro do *.CSV é transferido para o *.DAT, inserindo suas chaves (codINEP) no arquivo de índices. Finaliza-se o Buffer-Pool com a função finalizarBufferPool. Atualiza-se os arquivos de cabeçalho, marcando os arquivos como consistentes.

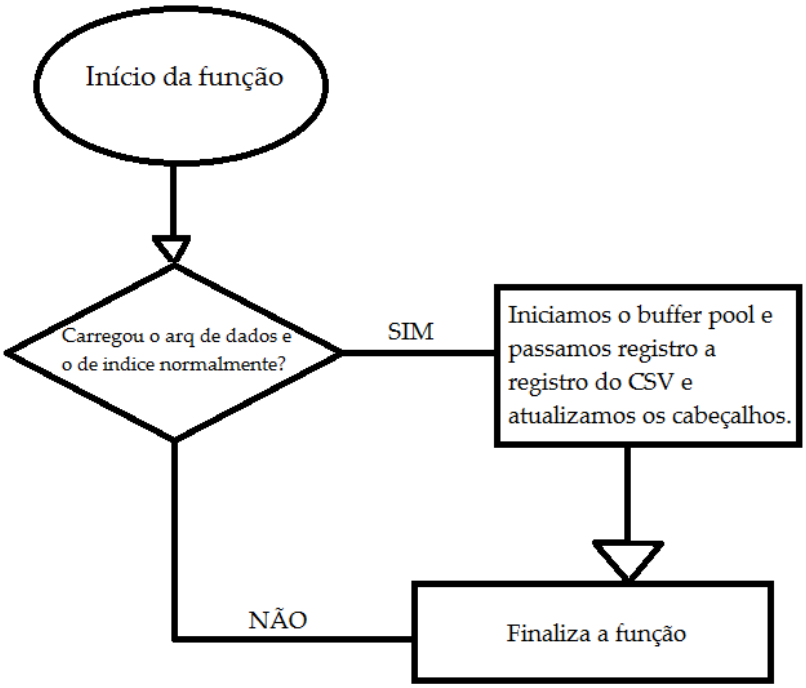

Figura 1 – Algoritmo em alto nível da transferirCSV.

A função inserirRegistro é chamada para a funcionalidade 11 e tem como objetivo inserir um registro no arquivo *.DAT. Primeiramente checa-se se os campos estão corretos e então os arquivos são abertos e checa-se suas consistências através dos cabeçalhos. Caso ambos estejam consistentes, torna-os inconsistentes para realizar modificações nestes. Inicia-se o Buffer-Pool, e caso não haja registros removidos da pilha (topo da pilha de registros logicamente removidos), insere-se ao final do arquivo de dados. Caso haja algum registro removido, insere-se nesta posição livre (reaproveitamento de espaço, como feito na primeira parte do trabalho prático).

Após a inserção no arquivo *.DAT, insere-se a chave (codINEP) no arquivo de índices. Por fim, o Buffer-Pool é finalizado e os cabeçalhos são atualizados, tornando os arquivos consistentes.

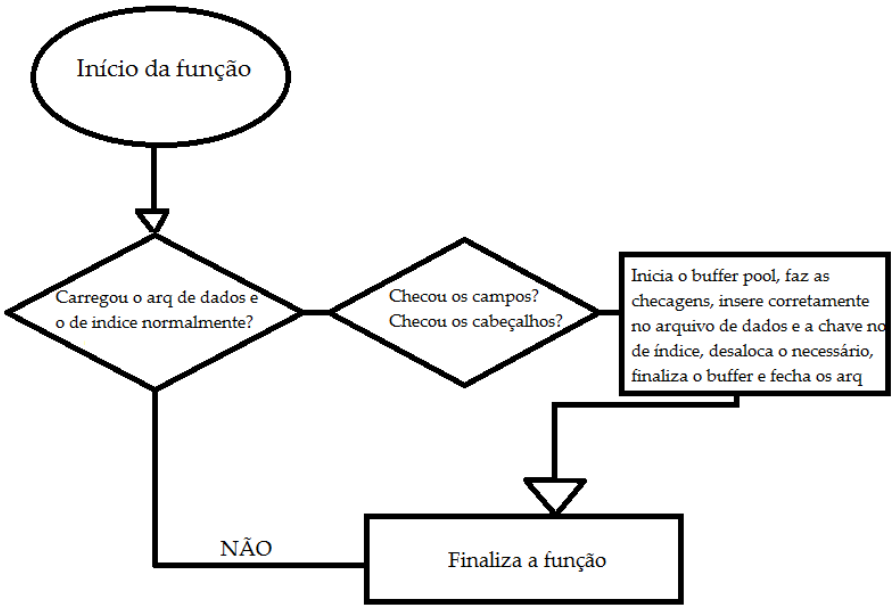

Figura 2 – Algoritmo em alto nível do inserirRegistro.

SEÇÃO 3:

A função recuperarPorChave é chamada para a funcionalidade 12 e tem como objetivo recuperar um registro do arquivo de dados com base no codINEP usando o índice árvore-B. Esta primeiramente abre o arquivo de dados e o arquivo de índice, e checa se seus status estão consistentes. Inicia-se o Buffer-Pool e então faz-se a busca na árvore-B com a função buscaRRN, que retorna um RRN de um registro no arquivo de dados. Finaliza-se o Buffer-Pool.

Agora é feito um *fseek* no arquivo de dados com o RRN que encontramos. Caso o RRN seja -1 ou o *fseek* falhe, o registro não existe. Caso contrário, lê o registro e imprime-o na saída padrão com a função imprimirRegistro. Fecha-se os arquivos de dados e de índice.

SEÇÃO 4:

A função removerRegistroChave é chamada para a funcionalidade 13 e tem como objetivo remover um registro do arquivo de dados através do codINEP buscado na árvore-B. Esta abre o arquivo de dados e o arquivo de índice, e checa se seus status estão consistentes. Como ambos serão modificados, torna-os inconsistentes.

Inicia-se então o Buffer-Pool e chama-se a função removerChave, para já removermos tal chave e recebermos o RRN dela. Caso não o registro não for encontrado, o algoritmo é terminado.

Caso seja um registro válido e existente no arquivo de dados, atribui-se a máscara de remoção, atualiza o topo da pilha de registros logicamente removidos e escreve as mudanças em disco. Finaliza-se o Buffer-Pool e torna os arquivos consistentes novamente.

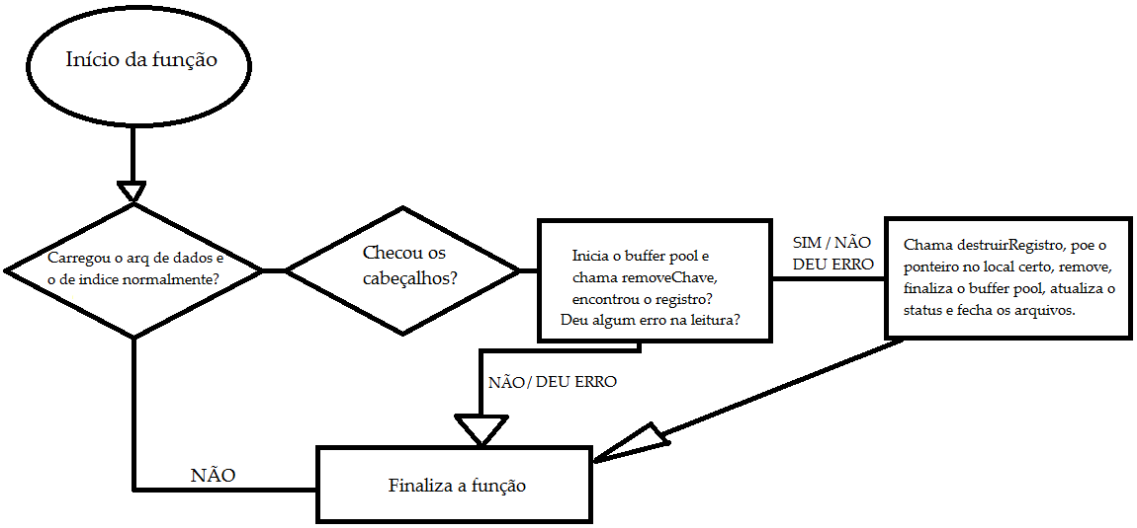

Figura 3 - Algoritmo em alto nível do removerRegistroChave.

SEÇÃO 5:

A função atualizarRegistroChave é chamada para a funcionalidade 14 e tem como objetivo realizar a atualização no arquivo de dados a partir da chave (codINEP) digitado pelo usuário e atualizar o arquivo de índice árvore-B com a nova chave.

Realiza-se primeiramente a validação dos campos. Abre-se os arquivos de dados e índice, torna-os inconsistentes e inicia o Buffer-Pool. Remove-se a chave antiga com removerChave, recebendo o RRN deste registro. Caso a chave foi encontrada e o registro é válido no arquivo de dados, faz-se a atualização do registro com os campos passados pelo usuário.

Por fim, escreve o registro no arquivo de dados, insere a nova chave no arquivo de índice, finaliza o Buffer-Pool e torna os arquivos consistentes novamente.

Referências Bibliográficas:

- Slides da professora Cristina Dutra de Aguiar Ciferri sobre Árvore-B.
- Slides *Gerenciamento de Buffer Pool*, por Anderson Chaves Carniel.

Disponível em <http://wiki.icmc.usp.br/index.php/SCC0215012018_Material_Didático_(cdac)> (acesso em junho de 2018)