UNIVERSIDADE DE SÃO PAULO
INSTITUTO DE CIÊNCIAS MATEMÁTICAS E DE COMPUTAÇÃO
Departamento de Ciências de Computação

Disciplina: **SCC0503 – Algoritmos e Estrutura de Dados II**
Professora: Elaine Parros Machado de Sousa
Estagiário PAE: Evandro Ortigossa

Trabalho 1 – Árvores-B

Instruções:

- **O trabalho deverá ser feito em duplas**;
- A entrega deve ser feita em **Atividade Trabalho Árvore-B** do Tidia até o dia **07/06/2020**, às 23:55h;
- Deve ser feito *upload* em um único arquivo compactado, contendo o **código fonte** e **um relatório pdf**. O nome do arquivo compactado deve ser formado pela sigla "Trab1_" concatenado a primeiro nome e último sobrenome do 1o aluno, conectado a primeiro nome e último sobrenome do 2o aluno. **Exemplo:** Trab1_ElaineSousa_EvandroOrgigossa.zip. **Somente 1** dos alunos do grupo deve fazer o *upload* da atividade!
- Faça um cabeçalho no início dos arquivos de código-fonte contendo as informações relevantes sobre seu trabalho, bem como nome completo e número USP dos integrantes do grupo, em comentário;
- Organize seu código utilizando comentários instrutivos;
- **No relatório**, explique: a lógica da sua solução, principalmente a estrutura lógica da implementação do nó da árvore-B e a solução de implementação; estrutura do arquivo de dados; funcionamento da interface; nomes dos arquivos de dados e de índice; e qualquer comentário que julgar necessário para entendimento da sua solução.

Enunciado:

Você foi contratado para desenvolver uma solução que auxilie o professor a organizar e armazenar os registros de seus alunos em disco. Para cada aluno, o professor deseja manter os seguintes atributos:

- Código do registro, representado pelo número **USP do aluno** (esse campo é chave, não existirão valores idênticos);
- **Nome** (*string* com o primeiro nome do aluno);
- **Sobrenome** (*string* com o sobrenome completo, por ex.: "da Silva";
- **Curso** (*string* com o código da disciplina);
- **Nota** (por ex.: 8,75).

Para tal, você deverá desenvolver um programa que permita ao professor as seguintes operações:

- Inserir um novo registro completo;
- Buscar um registro completo a partir da chave (NUSP);
- **(Opcional - Valendo 2 pontos extras)** Remover um registro a partir da chave.

O sistema deve ser baseado em um **arquivo de dados** para armazenamento dos registros completos e um arquivo de índice do tipo **Árvore-B.**

Arquivo de Dados:

Para a realização deste trabalho, você pode reaproveitar o desenvolvimento feito nas práticas anteriores para **arquivos de dados de tamanho fixo e campo fixo**. O arquivo de dados deverá ser ser ASCII (arquivo texto), não ordenado, com inserção no final do arquivo, sem quebras de linha. Poderá ser implementada a remoção lógica (opcional) dos registros de dados.
O arquivo de dados **não** deverá conter cabeçalho e deverá se chamar **dados.txt**.

Arquivo de Índice:

O arquivo de índice pode ser binário ou texto. A organização do índice utilizada no arquivo **deve** ser uma **árvore-B**. Note que os "ponteiros" para páginas armazenadas no arquivo de índice devem ser os **RRNs** da localização da página em questão neste arquivo. A árvore-B da sua implementação deve seguir os seguintes requisitos:

- Cada página da árvore, ou nó, deve ter o tamanho de um bloco de disco. Um bloco é uma sequência de *bytes* de tamanho fixo que pode variar entre 512 *bytes*, 4kB, 8kB, 16kB, 32kB etc. Vamos definir blocos com **4kB**, que é o tamanho mais utilizado atualmente;
- Os endereços (**RRNs**) devem ser do tipo long de 4 *bytes* (tanto os RRNs das próximas páginas da árvore, quanto os RRNs dos registros). "Ponteiros" nulos (não utilizados) devem ter valor "-1";
- Novas páginas devem ser colocadas no final do arquivo;
- O cabeçalho do arquivo de índice deve conter o RRN da página raiz;
- A implementação da remoção de chaves (e de páginas) da árvore-B é opcional (nota extra), embora seu desenvolvimento seja desafiador e um excelente aprendizado.

Observação 1: No material de aula, foi apresentada uma solução de implementação para os nós da Árvore-B, usando *structs* e vetores. Abaixo, é apresentada uma outra solução. O grupo deve escolher uma abordagem de implementação (que pode ser diferente das apresentadas) e documentar muito bem no relatório a solução adotada.

Na solução apresentada aqui, como os campos do índice possuem tamanho fixo, tudo deve ser armazenado sem caracteres delimitadores e nem quebras de linha, como no exemplo abaixo, de uma página de árvore-B:

$$O_P<C_RO_R>O_P<C_RO_R>O_P \ldots O_P<C_RO_R>O_P<C_RO_R>O_P$$

Cada O_P representa o RRN para outra página da árvore. Cada par $<C_RO_R>$ representa o par ordenado composto pela chave de um registro e o RRN deste registro no arquivo de dados. Note que os símbolos "<" e ">" foram colocados nesse exemplo apenas para facilitar a leitura das informações e não devem constar no arquivo. As páginas devem ser armazenadas sem delimitadores. O exemplo acima representa uma única página da árvore-B, a segunda página deve começar imediatamente após o término da primeira página (e assim por diante), sem caracteres delimitadores nem quebras de linha.
Seu arquivo de índice deve se chamar **index.dat**.

Observação 2: Lembrando dos fundamentos de arquivos, apresentados na Aula 1, observe que a página (bloco de disco), é a unidade mínima de alocação de espaço e de leitura de dados do disco para RAM (ver pg. 22 do pdf da Aula 1). A ideia da Árvore-B é que cada nó ocupe exatamente uma página, para que todo nó possa ser lido e

carregado em um único *seek*. Para que isso funcione, é necessário adequar os dados armazenados no arquivo de índice ao tamanho de página (4kB, no nosso caso). Isso significa que, por exemplo, deverá ser alocada uma página inteira (de 4kB) para armazenar o cabeçalho (página 0), de modo cada nó seja, posteriormente, armazenado por completo numa nova página (ex: página 1 para nó raiz inicial). A mesma lógica deve ser aplicada na definição da quantidade de chaves por nó, ou seja, a soma de tamanhos de chaves + RRNs + info adicional (ex: contador de elementos) não deve ultrapassar os 4KB, mesmo que sobre um pouco de espaço na página. Por isso: planeje bem os tamanhos dos campos dos registros de índice para maximizar o uso de espaço.

IMPORTANTE: Note que, independente da solução adotada para implementação do nó (página) da árvore, cada chave e respectivos RRNs representam uma quantidade de informação de tamanho bastante reduzido. Como vamos utilizar nós de tamanho real, ou seja, do tamanho de um bloco de disco, será necessário inserir muitos registros para formar um índice em árvore-B com diversas páginas (nós). Logo, o grupo também deverá desenvolver um gerador automático de dados para fazer o povoamento inicial da árvore (**no mínimo 2 níveis da árvore**). Como sugestão, vocês podem gerar as chaves de registro (número USP) de modo sequencial, a nota pode ser gerada aleatoriamente e os campos textuais (nome, sobrenome e curso) podem ser gerados utilizando algum gerador disponível na Internet (não é necessário desenvolver esse gerador, o grupo é livre para buscar e utilizar uma solução ou biblioteca pronta, mas não deixe de documentar como essa solução deve ser operada por nós, que a princípio, não sabemos qual é nem como utilizar).

Interação com o usuário:

Seu programa deve permitir interação com o usuário pelo console (modo texto), e deve oferecer o seguinte conjunto mínimo de operações:

1) **Inicialização: criação** dos arquivos de dados e de índice vazios, e **alimentação** de ambos utilizando os geradores automáticos de dados. A inserção no arquivo de dados é sequencial, no final do arquivo, sem quebras de linhas, com inserção correspondente no arquivo de índice, usando o procedimento de **Inserção em Árvore-B** visto em aula (com inserção de registros no final do arquivo). Observe que **arquivo de dados e arquivo de dados são criados juntos e devem crescer juntos**.
   A partir da primeira inicialização, para as demais interações com o sistema, supõe-se que ambos os arquivos já tenham sido preenchidos com uma quantidade (arbitrária) de registros. Assim, seu programa deverá buscar os dados já gravados no **arquivo de dados** por meio de acessos ao **arquivo de índice.** Para reduzir acessos a disco, é possível manter o nó raiz sempre em RAM (somente o nó raiz), desde que seja implementado o devido controle de consistência entre o que está em RAM e o que está no arquivo de índice (similar ao que é necessário para arquivos de índices primários e secundários carregados para memória). Os demais nós da árvore só serão carregados em RAM quando precisarem ser acessados para busca/inserção e/ou remoção;

2) **Inserção de registro:** O usuário deve ser capaz de inserir um novo registro de dados. O programa deve ler os seguintes campos: chave (número USP), nome, sobrenome, curso e nota. Essa operação implica inserção nos arquivos de dado e de índice. Lembrando que o NUSP é chave e não pode ser repetido;

3) **Recuperação (busca) de registro:** O usuário deve ser capaz de buscar um registro de dados completo. Caso não exista, seu programa deve informar tal fato ao usuário. Para buscar um registro, seu programa deve solicitar como entrada ao usuário somente a chave (NUSP). Todos os dados do registro recuperado devem ser impressos na tela de modo formatado, isto é, os campos devem ser separados e identificados (caso exista). **A busca deve, obrigatoriamente, ser feita via índice**;

4) **Finalizar a execução:** o usuário deve ser capaz de encerrar a execução do programa. Ao final da execução, todos os arquivos devem ser fechados e toda a memória alocada deve ser liberada pelo seu programa.

Caso você implemente a remoção **opcional**, forneça ao usuário a capacidade de remover um registro. Caso não exista o registro, seu programa deve informar tal fato ao usuário. Para remover um registro, seu programa deve solicitar como entrada ao usuário somente o campo chave. A remoção de um registro pode ser feita de **modo lógico**, utilizando o caracter especial "*" no primeiro *byte* do registro apagado no arquivo de dados. No arquivo de índice, deve ser implementado o procedimento de Remoção em Árvore-B (a remoção de páginas pode ser **lógica**, ou seja, só com marcador de página removida).

Observações:

1. Implemente seu programa utilizando a linguagem **C padrão ANSI**. Caso estiver usando o compilador GCC, utilize a *tag* -ansi. Utilize bibliotecas da linguagem **C ANSI**;
2. Você pode utilizar os **seus** códigos desenvolvidos nas práticas de laboratório;
3. A clareza da sua implementação será avaliada;
4. Use comentários relevantes em seu código, pois serão avaliados;
5. Vazamento de memória, referência a valores de variáveis não inicializadas e outros problemas de programação serão levados em conta na avaliação;
6. Você tem tempo suficiente para desenvolver e testar todo o trabalho... **não deixe para a última hora**;
7. Variáveis globais não são recomendadas;

8. O **PAE (Evandro)** ficará online durante os horários de aula (email ou chat do hangout - evortigosa@gmail.com) para apoio ao desenvolvimento e dúvidas. **IMPORTANTE:** o atendimento só será feito nesses horários - são quase 4 horas de atendimento por semana até a data de entrega. Lembrando:
   - **3a - das 21h às 22:40h**
   - **6a - das 19h às 21:40h**
   - **De 15/05 até 05/06.**

9. Lembrem-se da política sobre plágios que foi discutida na aula de apresentação da disciplina.

DICA para teste e debug: para testar se os procedimentos estão funcionando corretamente, faça uma primeira versão do programa com páginas pequenas e consequentemente com poucas chaves. Monte casos de testes com poucos dados que permitam testar todos os procedimentos e casos especiais (ex: quebras de nó, aumento na altura da árvore, busca por chave inexistente, etc...). Isso facilitará teste e debug. Depois de tudo funcionando corretamente, é só alterar o tamanho da página para 4KB, realizar os ajustes necessários e testar para cenário real.