Universidade Federal de São Carlos
Bacharelado em Ciência da Computação
Estruturas de Dados II
Prof. Tiago A. Almeida <talmeida@ufscar.br>

TRABALHO 02 - ÁRVORES-B

Atenção

1. **Prazo de entrega: 11/11/2018 – 23h55 (submissão online)**
2. **Sistema de submissão:** `http://judge.sor.ufscar.br/ed2/`
3. A atividade deverá ser realizada individualmente

Indexação usando árvores-B

O sistema de cadastro de peças para computador da *CaBum!*, UfsBum!, foi um sucesso e logo diversos clientes decidiram aderir ao sistema. Contudo, com o crescimento acelerado do arquivo de dados, as consultas começaram a ficar lentas e, em consequência, seu programa vem perdendo credibilidade e recebendo uma enxurrada de reclamações dos usuários.

Após analisar o cenário atual, concluiu-se que o uso de índices simples não é mais viável para realizar buscas no arquivo de dados, e que a melhor saída é usar índices de árvores-B para aumentar a eficiência do sistema.

Lembrando, cada produto (registro no arquivo de dados) é composto pelos seguintes campos:

- *Código*: composição de letras maiúsculas das duas primeiras letras do nome do produto (modelo), seguido das duas primeiras letras do nome da marca, do dia e mês da data de registro do produto na loja (com dois dígitos cada), e o ano de lançamento (últimos dois dígitos). Ex: `GEMS110917`. Esse campo é a *chave primária*, portanto, não poderá existir outro valor idêntico na base de dados;
- *Nome do produto ou modelo* (nome ou modelo pelo qual os usuários conhecem o produto, ex: `GeForce GTX 1080 TI ARMOR 11G OC`);
- *Marca* (nome da empresa que produz o produto, ex: `NVidia`);
- *Data de registro* (data no formato `DD/MM/AAAA`, ex: 24/09/2018);
- *Ano de lançamento* (inteiro com 2 dígitos, representando o ano de lançamento do produto, ex: `17`);
- *Preço-base* (valor de ponto flutuante com precisão de dois dígitos referente ao preço-base do produto, ex: `4139.41`);

- *Desconto* (inteiro com 3 dígitos, contendo a porcentagem de desconto que será abatida no preço-base durante a temporada de vendas, ex: `040` – nesse caso, o produto em questão ficaria com o preço final igual a 2483.65.
- *Categorias* (campo multi-valorado separado pelo caractere '|' se houver mais de uma categoria, ex: `PLACA DE VIDEO|GAMER|MULTIMIDIA`);

Garantidamente, nenhum campo de texto receberá caractere acentuado.

Tarefa

Desenvolva um programa que permita ao usuário manter uma base de dados de produtos. O programa deverá permitir:

1. Inserir um novo produto;
2. Modificar o campo **desconto** de um produto a partir da chave primária;
3. Buscar produtos a partir:
   1) da chave primária
   2) marca e nome do produto ou modelo.
4. Listar todos os produtos da base de dados ordenados por:
   1) impressão pré-ordem da árvore-B primária
   2) impressão pré-ordem da árvore-B secundária
5. Visualizar o arquivo de Dados;
6. Visualizar o arquivo de Índice Primário;
7. Visualizar o arquivo de Índice Secundário.

Novamente, nenhum arquivo ficará salvo em disco. O arquivo de dados e os de índices serão simulados em *strings* e os índices serão sempre criados na inicialização do programa e manipulados em memória RAM até o término da execução. Suponha que há espaço suficiente em memória RAM para todas as operações.

Arquivo de dados

O arquivo de dados deve ser ASCII (arquivo texto), organizado em registros de tamanho fixo de 192 bytes. Os campos *nome do produto ou modelo*, *marca* e *categorias* devem ser de tamanho variável. Os demais campos devem ser de tamanho fixo: *código* (10 bytes), *data de registro* (10 bytes), *preço-base* (7 bytes), *ano de lançamento* (2 bytes) e *desconto* (3 bytes). A soma de bytes dos campos fornecidos (incluindo os delimitadores necessários) nunca poderá ultrapassar 192 bytes. Os campos do registro devem ser separados pelo caractere delimitador @ (arroba). Cada registro terá 7 delimitadores, mais 32 bytes ocupados pelos campos de tamanho fixo. Você precisará garantir que os demais campos juntos ocupem um máximo de 153 bytes. Caso o registro tenha menos de 192 bytes, o espaço adicional deve ser marcado com o caractere # de forma a completar os 192 bytes. Para evitar que o registro exceda 192 bytes, os campos variáveis devem ocupar no máximo 51 bytes.

```
PLMS271017@PLACA DE VIDEO NVIDIA 1080 TI ARMOR 11G OC@MSI@27/10
/2017@17@4869.90@015@PLACA DE VIDEO|HARDWARE@##################
############################################################
###TEGA150418@TECLADO GAMER RAZER CYNOSA CHROMA MEMBRANA - US@R
AZER@15/04/2018@18@418.71@010@TECLADO|PERIFERICO@##############
############################################################
######SSWD110117@SSD 2.5 1TB SATA III 6GB/S - WDS100T2B0A@WD@11
/01/2018@17@1823.41@010@ARMAZENAMENTO|HARDWARE@################
############################################################
#########G.G.150318@G.SKILL TRIDENT Z RGB 32GB (4x8) 3200MHZ DD
R4 CL 16@G.SKILL@15/03/2018@18@2716.90@015@MEMORIA RAM|HARDWARE
@##########################################################
############MORA240418@MOUSE DEATHADDER ELITE CHROMA MECANICO 1
6.000 DPI@RAZER@24/04/2018@18@0503.41@007@MOUSE|PERIFERICO@####
############################################################
##############################
```

Note que não há quebras de linhas no arquivo (elas foram inseridas aqui apenas para exemplificar a sequência de registros).

Instruções para as operações com os registros:

- **Inserção:** cada produto deverá ser inserido no final do arquivo de dados e atualizado nos índices.
- **Atualização:** o único campo alterável é o de *Desconto*. O registro deverá ser localizado acessando o índice primário e o desconto deverá ser atualizado no registro na mesma posição em que está (não deve ser feita remoção seguida de inserção). Note que o campo de *Desconto* sempre terá 3 dígitos.

Arquivo de Índices

No cenário atual, **os índices não cabem em RAM** e, portanto, para simular essa situação, dois "*Arquivos de Índices*: `iprimary` e `ibrand`" deverão ser criados na inicialização do programa e manipulados em RAM até o encerramento da aplicação.

Para ambas as árvores, as ordens serão informadas pelo usuário e **a promoção deverá ser sempre pelo sucessor imediato** (menor chave da sub-árvore direita). Todo novo nó criado deverá ser inserido no final do respectivo arquivo de índice.

1. `iprimary`: índice primário (Árvore-B), contendo as chaves primárias e os RRNs dos registros no arquivo de dados. Cada registro da árvore primária é composto por:

   - 3 bytes para a quantidade de chaves;
   - (Ordem - 1) * (10 bytes de chave primária + 4 bytes para o RRN do arquivo de dados). Para as chaves não usadas, preencha todos os bytes com `'#'`;
   - 1 byte para indicar se o nó é folha `'F'` ou não `'N'`;
   - Por fim, uma quantia de (ordem * 3 bytes) para indicar os RRNs dos nós descendentes do nó atual. Note que esse RRN se refere ao próprio arquivo de índice primário. Utilize `'***'` para indicar que aquela posição do vetor de descendentes é nula.

   Exemplo da representação da árvore B de ordem 3, após a inserção das chaves iniciadas com: LEWA, MEKO e BO2K (nesta ordem).

Figura 1: Árvore-B de ordem 3 após inserção do primeiro registro

Em disco, o arquivo de índice primário seria:

```
001LEWA0412000000##############F*********
```

Figura 2: Árvore-B de ordem 3 após inserção do segundo registro

Em disco:

```
002LEWA0412000000MEKO1401180001F*********
```

Note que ao inserir a chave BO2K irá ocorrer um *overflow* na raíz, sendo necessário criar então, 2 novos nós (de RRN 1 e 2 respectivamente). O primeiro, será para a redistribuição de chaves, e o segundo para receber a promoção de chave que será a nova raíz. Visualmente falando:

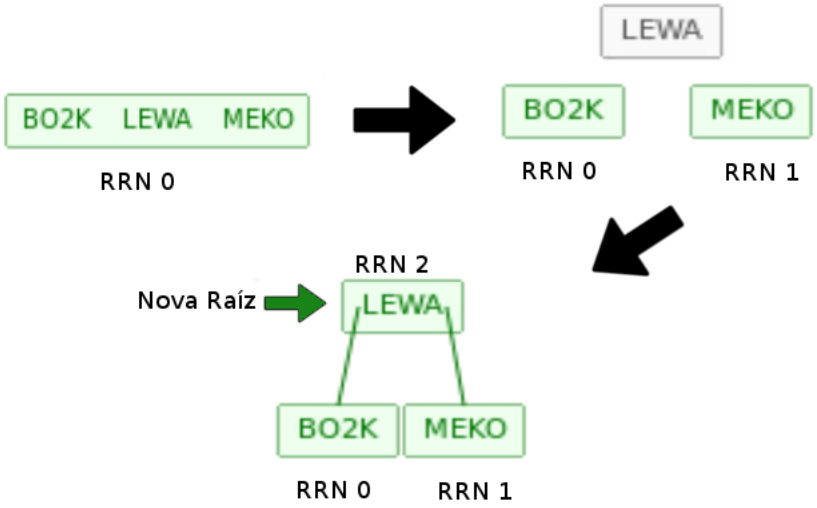

Figura 3: Árvore-B de ordem 3 após inserção do terceiro registro

```
001BO2K2411030002##############F*********
001MEKO1401180001##############F*********
001LEWA0412000000##############N000001***
```

Note que, aqui também não há quebras de linhas no arquivo. Elas foram inseridas apenas para exemplificar a sequência de registros.

2. `ibrand`: índice secundário (Árvore-B), contendo suas chaves composta por: (*i*) a chave primária do respectivo registro e (*ii*) o nome da marca concatenado com '`$`' e o nome do produto ou modelo, sendo que as chaves devem ser ordenadas pela ordem lexicográfica da segunda *string* (*ii*). Cada registro da árvore secundária é composto por:

   - 3 bytes para a quantidade de chaves,
   - (Ordem - 1) * (10 bytes de chave primária + 101 bytes da *string*). Para as chaves não usadas, preencha todos os 111 bytes com '`#`'.
   - 1 byte para indicar se o nó é uma folha '`F`' ou não '`N`';
   - Por fim, uma quantia de (ordem * 3 bytes) para indicar os RRNs dos nós descendentes. Note que assim como no índice primário, esse RRN se refere ao próprio arquivo de índice secundário. Utilize '`***`' para indicar que aquela posição do vetor de descendentes é `null`.

   Exemplo de arquivo de índice secundário representado por uma árvore-B de ordem 4, após a inserção das chaves na ordem: '`WAHD$LENEW HD 1TB SATA`' (`WALE`), '`KONAN PIECES$MEMORIA RAM 16GB`' (`KOME`) e '`2K PRICES$BOLD MONITOR 42 LED`' (`2KBO`).

Figura 4: Índice secundário estruturado em Árvore-B de ordem 4.

```
003BO2K2411032K PRICES$BOLD MONITOR 42 LED####################
##################################################MEKO140118KO
NAN PIECES$MEMORIA RAM 16GB DDR4##############################
###################################LEWA041200WAHD$LENEW HD 1TB
 SATA########################################################
######################F***********
```

Novamente, não há quebras de linhas no arquivo. Elas foram inseridas apenas para exemplificar a sequência de registros.

É terminantemente proibido manter uma cópia dos índices inteiros em TADs. A única informação que você deverá manter todo tempo em memória, é o RRN da raíz de cada árvore. Assuma que um nó do índice corresponde a uma página e, portanto, cabe no *buffer* de memória. Dessa forma, trabalhe com apenas a menor quantidade de nós necessários das árvores por vez, pois isso implica em reduzir a quantidade de *seeks* e de informação transferida entre as memórias primária e secundária.

Deverá ser desenvolvida uma rotina para a criação de cada índice. Os índices serão criados e manipulados sempre utilizando os pseudo-arquivos de índices. Note que o ideal é que a árvore-B `iprimary` seja a primeira a ser criada.

Para que isso funcione corretamente, o programa, ao iniciar precisa realizar os seguintes passos:

1. Perguntar ao usuário se ele deseja informar um arquivo de dados:
   - Se sim: recebe o arquivo inteiro e armazena no vetor `ARQUIVO`.
   - Se não: considere que o arquivo está vazio.
2. Inicializar as estruturas de dados dos índices:
   - Solicitar as ordens m e n das duas árvores e criar a estrutura dos índices.

Interação com o usuário

O programa deve permitir interação com o usuário pelo console/terminal (modo texto) via menu.

A primeira pergunta do sistema deverá ser pela existência ou não do arquivo de dados. Se existir, deve ler o arquivo e armazenar no vetor `ARQUIVO`. Em seguida, o sistema deverá perguntar pelas ordens m e n das árvores-B usadas para indexação.

As seguintes operações devem ser fornecidas (nessa ordem):

1. **Cadastro.** O usuário deve ser capaz de inserir um novo produto. Seu programa deve ler os seguintes campos (nessa ordem): **nome do produto ou modelo**, **marca**, **data de registro**, **ano de lançamento**, **preço-base**, **desconto** e **categorias**. Note que a chave **não é** inserida pelo usuário, você precisa gerar a chave para gravá-la nos índices. Garantidamente, os campos serão fornecidos de maneira regular, não sendo necessário um pré-processamento da entrada. Se um novo registro possuir a chave gerada igual a de um outro registro já presente no arquivo

de dados, a seguinte mensagem de erro deverá ser impressa: "`ERRO: Já existe um registro com a chave primária AAAA999999.\n`", onde `AAAA999999` corresponde à chave primária do registro que está sendo inserido e `\n` indica um pulo de linha após a impressão da frase.

2. **Alteração.** O usuário deve ser capaz de alterar o desconto de um produto informando a sua chave primária. Caso ele não exista, seu programa deverá exibir a mensagem "`Registro não encontrado!\n`" e retornar ao menu. Caso o registro seja encontrado, certifique-se de que o novo valor informado está dentro dos padrões (*i.e.*, 3 bytes, com o valor entre 000 e 100) e, nesse caso, altere o valor do campo diretamente no arquivo de dados. Caso contrário, exiba a mensagem "`Campo inválido!\n`" e solicite a digitação novamente. Ao final da operação, imprima "`OPERACAO REALIZADA COM SUCESSO!\n`" ou "`FALHA AO REALIZAR OPERACAO!\n`".

3. **Busca.** O usuário deve ser capaz de buscar por um produto:

   - **1. por código:** solicitar ao usuário a chave primária. Caso o produto não exista, seu programa deve exibir a mensagem "`Registro nao encontrado!`" e retornar ao menu principal. Caso o produto exista, todos os dados deverão ser impressos na tela de forma formatada, exibindo os campos na mesma ordem de inserção. Em ambos os casos, seu programa deverá imprimir o caminho percorrido na busca exibindo as chaves contidas nos nós percorridos. **Na última linha do caminho percorrido, adicione uma quebra de linha adicional.**

     Por exemplo, considere a seguinte árvore-B de ordem 3 resultante da inserção das seguintes chaves: `LEKO041200`, `MEVA140118`, `BOBE241103`, `THSE271000`, `HAVA160314`, `CABE180614`, `COUB011221`, `GRNA120803`, `BASQ240318` e `XCBE201105` (Figura 5).

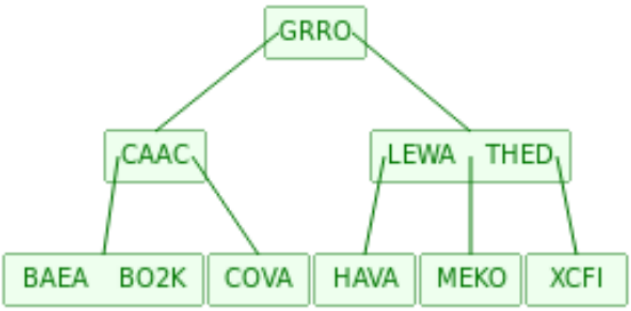

Figura 5: Índice primário estruturado em Árvore-B de ordem 3.

A busca pela chave `MEKO140118` e `CAAC999999` retornará:

```
*********************************BUSCAR*********************************
Busca por MEKO140118. Nos percorridos:
GRRO120803
LEWA041200, THED271000
MEKO140118
```

```
MEKO140118
MEMORIA RAM 16GB
KONAN PIECES
14/01/2017
18
1348.50
MEMORIA, HARDWARE
**********BUSCAR**********
Busca por CAAC999999. Nos percorridos:
GRRO120803
CAAC180614
COVA011221

Registro(s) nao encontrado!
```

- **2. por marca e nome do produto ou modelo:** solicitar ao usuário o nome da marca seguido do nome do produto ou modelo. Caso nenhum produto tenha sido encontrado, o programa deve exibir a mensagem "`Registro(s) nao encontrado!`" e retornar ao menu principal. Exemplo de utilização para as buscas: {'`2K PRICES`','`BOLD MONITOR 42 LED`'} e {'`VALENTIAN`','`HARD DRIVE 2TB SATA`'}

```
**********BUSCAR**********
Busca por 2K PRICES$BOLD MONITOR 42 LED.
Nos percorridos:
EAR STUFF$BATTLE HEADPHONE, KONAN PIECES$MEMORIA RAM16GB,
    VALENTIAN$HARD DRIVE 2TB SATA
2K PRICES$BOLD MONITOR 42 LED, ACER$CABUM EDITION MONITOR 24 LED

BO2K241103
BOLD MONITOR 42 LED
2K PRICES
24/11/2017
03
2999.99
MONITOR, PERIFERICO, GAMER
**********BUSCAR**********
Busca por VALENTIAN$HARD DRIVE 3TB SATA.
Nos percorridos:
EAR STUFF$BATTLE HEADPHONE, KONAN PIECES$MEMORIA RAM 16GB,
    VALENTIAN$HARD DRIVE 2TB SATA

Registro(s) nao encontrado!
```

4. **Listagem.** O sistema deverá oferecer as seguintes opções de listagem:

   - **1. árvore-B primária:** imprime a `iprimary` (somente o campo de chave primária) usando varredura **pré-ordem**. Imprimir um nó por linha, começando pelo nível da árvore em que se encontra o nó (a partir da raiz: nível 1) seguido da chave. Caso não haja nenhum registro, imprima a mensagem de `'Registro(s) nao encontrado!'`. Por exemplo, considere a árvore-B apresentada na Figura 5, a sua listagem resultaria em:

```
*********************************LISTAR*********************************
1 - GRRO120803
2 - CAAC180614
3 - BAEA240318, BO2K241103
3 - COVA011221
2 - LEWA041200, THED271000
3 - HAVA160314
3 - MEKO140118
3 - XCFI201105
```

   - **2. Árvore-B secundária:** realiza uma travessia **em ordem**, exibindo todas as marcas e produtos na ordem lexicográfica das marcas e nomes dos produtos. Exemplo:

```
*********************************LISTAR*********************************
2K PRICES----------------------- BOLD MONITOR 42 LED-----------------
ACER---------------------------- CABUM EDITION MONITOR 24 LED--------
EAR STUFF----------------------- BATTLE HEADPHONE--------------------
ED2 ALLIANCE-------------------- THE PANCADAO MONITOR 50 LED---------
FIRAGA PIECES------------------- XC COOLER AMD INTEL-----------------
KONAN PIECES-------------------- MEMORIA RAM 16GB--------------------
ROLLING RICK-------------------- GRNANEOVNE VPGEYUUOI HD 1TB---------
VALENTIAN----------------------- COOLER HYPER VENT LED VERMELHOR-----
VALENTIAN----------------------- HARD DRIVE 2TB SATA-----------------
WAHD---------------------------- LENEW HD 1TB SATA--------------------
```

     Note que o número de tracejados foi diminuído devido ao tamanho da margem do PDF, porém, cada linha é composta por: 50 caracteres (marca) + 1 espaço em branco + 50 caracteres (nome do produto ou modelo) + retorno de linha.

5. **Imprimir Arquivo de dados.** Imprime o Arquivos de dados, caso esteja vazio apresente a mensagem "`Arquivo vazio!`".

6. **Imprimir Índice Primário.** Imprime o Arquivo de índice primário, sendo **um nó da árvore por linha**. Caso esteja vazio apresente a mensagem "`Arquivo vazio!`".

7. **Imprimir Secundário.** Imprime o Arquivo de índice secundário, sendo **um nó da árvore por linha**. Caso esteja vazio apresente a mensagem "`Arquivo vazio!`".

8. **Finalizar.** Encerra a execução do programa. Ao final da execução, libere toda a memória alocada pelo programa caso ainda possua alguma.

Implementação

Implemente suas funções utilizando o código-base fornecido. **Não é permitido modificar os trechos de código fornecidos ou as estruturas já definidas**. Ao imprimir alguma informação para o usuário, utilize as constantes definidas. Ao imprimir um registro, utilize a função `exibir_registro()`.

Tenha atenção redobrada ao implementar as operações de busca e listagem da árvore-B. Atente-se às quebras de linhas requeridas e não adicione espaços em branco após o último caractere imprimível. A saída deverá ser exata para não dar conflito com o `Judge`. Em caso de dúvidas, examine os casos de teste.

Você deve criar obrigatoriamente as seguintes funcionalidades:

- Criar o índice primário: deve construir o índice primário a partir do arquivo de dados e da ordem m informada na inicialização do programa;
- Criar o índice secundário: deve construir o índice secundário a partir do arquivo de dados e da ordem n informada na inicialização do programa;
- Inserir um registro: modificar o arquivo de dados e os arquivos de índices.
- Buscar por registros: buscar por registros pela chave primária ou secundária.
- Alterar um registro: modificar o arquivo de dados.
- Listar registros: listar as árvores-B.
- Finalizar: deverá ser chamada ao encerrar o programa e liberar toda a memória alocada.

Utilizar a linguagem `ANSI C`.

Dicas

- Você nunca deve perder a referência do começo dos arquivos, então não é recomendável percorrer as *strings* diretamente pelos ponteiros `ARQUIVO`, `ARQUIVO_IP` e `ARQUIVO_IS`. Um comando equivalente a `fseek(f, 192, SEEK_SET)` é `char *p = ARQUIVO + 192`.
- Diferentemente do `fscanf`, o `sscanf` não movimenta automaticamente o ponteiro após a leitura.
- O `sprintf` adiciona automaticamente o caractere `\0` no final da *string* escrita. Em alguns casos você precisará sobrescrever a posição manualmente. Você também pode utilizar o comando `strncpy` para escrever em *strings*, esse comando, diferentemente do `sprintf`, não adiciona o caractere nulo no final.

- Ao utilizar o comando `strcpy` certifique-se que a *string* destinatária possui tamanho maior ou igual que a de origem, caso contrário poderá realizar escrita em espaço inapropriado da memória. Como alternativa use a `strncpy`.
- Não é possível retornar mais de um valor diretamente em C, mas a linguagem disponibiliza a criação de `structs` e também a passagem por referência para simular tal recurso.
- A função `strtok` permite navegar nas *substrings* de uma certa *string* dado(s) o(s) delimitador(es). Porém, tenha em mente que ela deve ser usada em uma cópia da *string* original, pois ela modifica o primeiro argumento.
- Utilize ferramentas de depuração, tais como `GDB` e `Valgrind`, para encontrar erros específicos e aumentar a sua produtividade.

CUIDADOS

Leia atentamente os itens a seguir.

1. O projeto deverá ser submetido no `Sharif Judge` em um único arquivo com o nome `{RA}_ED2_T02.c`, sendo `{RA}` correspondente ao número do seu RA;
2. Não utilize acentos nos nomes de arquivos;
3. Dúvidas conceituais deverão ser colocadas nos horários de atendimento. Dificuldades em implementação, consultar os monitores da disciplina nos horários estabelecidos;
4. **Documentação:** inclua cabeçalho, comentários e indentação no programa;
5. **Erros de compilação:** nota **zero** no trabalho;
6. **Tentativa de fraude:** nota **zero na média** para todos os envolvidos.