Universidade Federal de São Carlos
Bacharelado em Ciência da Computação
Organização e Recuperação da Informação
Prof. Tiago A. Almeida <talmeida@ufscar.br>

TRABALHO 02 - ÁRVORES-B

Atenção

1. **Prazo de entrega: 10/11/2019 às 23h55 (submissão online)**
2. **Sistema de submissão:** `http://judge.sor.ufscar.br/ori/`
3. A atividade deverá ser realizada individualmente

Indexação usando árvores-B

O `UFSCarona` foi um sucesso e logo outras universidades 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 CARONA (registro no arquivo de dados) é composto pelos seguintes campos:

- *Código*: composição de letras maiúsculas da primeira letra do nome do motorista, seguida das três primeiras letras da placa do veículo, seguidas do dia e mês da data do trajeto (com dois dígitos cada) e a hora da partida (dois primeiros dígitos). Ex: `AFGZ240907`. Esse campo é a *chave primária*, portanto, não poderá existir outro valor idêntico na base de dados;
- *Nome do motorista* (primeiro nome, pelo qual os caronistas entrarão em contato, ex: `ALEXANDRE`);
- *Gênero* (gênero do motorista, Masculino (`M`) ou Feminino (`F`));
- *Data de nascimento* (data no formato `DD/MM/AAAA`, ex: `24/09/1999`);
- *Celular* (número de contato do motorista com DDD, ex: `(15) 99815-1234`);
- *Veículo* (veículo que será usado no trajeto, ex: `BELINA`);
- *Placa* (placa do veículo, ex: `FGZ-1234`).

- *Trajeto* (campo multi-valorado separado pelo caractere '|'. O primeiro valor será sempre a origem e o último o destino, porém podem ter um ou mais trajetos no meio, ex: `AFONSO VERGUEIRO|PANNUNZIO|UFSCAR`);
- *Data do trajeto* (data no formato `DD/MM/AA`, ex: `24/09/19`);
- *Hora da partida* (hora no formato `HH:MM`, ex: `07:30`);
- *Valor* (preço da carona no formato `999.99`, ex: `004.50`);
- *Vagas* (quantidade de vagas disponíveis, ex: `3`).

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 nova carona;
2. Modificar o campo **vagas** a partir da chave primária;
3. Buscar caronas a partir:
   1) da chave primária
   2) do destino, data e hora.
4. Listar todas as caronas da base de dados ordenadas 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 256 bytes (256 caracteres). Os campos *nome do motorista*, *modelo do carro* e *trajeto* devem ser de tamanho variável. Os demais campos devem ser de tamanho fixo: *código* (10 bytes), *gênero* (1 byte), *data de nascimento* (10 bytes), *celular* (15 bytes), *placa do veículo* (8 bytes), *data* (8 bytes) e *hora* (5 bytes) da carona, *valor* (6 bytes) e por fim *vagas disponíveis* (1 byte). A soma de bytes dos campos fornecidos (incluindo os delimitadores necessários) nunca poderá ultrapassar 256 bytes. Os campos do registro devem ser separados pelo caractere delimitador `@` (arroba). Cada registro terá 12 delimitadores, mais 64 bytes ocupados pelos campos de tamanho fixo. Você precisará garantir que os demais campos juntos ocupem um máximo de 180 bytes. Caso o registro tenha menos de 256 bytes, o espaço adicional deve ser marcado com o caractere ♯ de forma a completar os 256 bytes. Para evitar que o registro exceda 256 bytes, os campos variáveis devem ocupar no máximo 180 bytes. Exemplo:

```
GABRIEL AUGUSTO@M@22/07/1995@(11) 99542-4321@CORSA@ABC-4321@SO
ROCABA|SAO ROQUE|COTIA|SAO PAULO@30/10/19@15:00@020.00@4@#####
############################################################
############################################################
#######LETICIA FERREIRA DA SILVA@F@14/04/1990@(16) 95435-2134@
ONIX@CAC-1010@SAO PAULO|SAO JOSE DOS CAMPOS|TAUBATE|LORENA|VOL
TA REDONDA|RIO DE JANEIRO@12/10/19@07:00@150.00@3@############
############################################################
###############ALEXANDRE@M@24/09/1999@(15) 99815-1234@BELINA@F
GZ-1234@AFONSO VERGUEIRO|PANNUNZIO|UFSCAR@24/09/19@07:30@004.5
0@3@##########################################################
############################################################
#######################ROBERT DOWNEY JUNIOR F :(@M@04/04/1965@
(11) 98754-1252@E-TRON GT@MAN-6969@LAS VEGAS|LOS ANGELES@25/04
/20@10:15@599.90@1@###########################################
############################################################
###############################
```

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 oferta de carona deverá ser inserida no final do arquivo de dados e atualizado nos índices.
- **Atualização:** o único campo alterável é o de *Vagas*. O registro deverá ser localizado acessando o índice primário e campo deverá ser atualizado no registro na mesma posição em que está (não deve ser feita remoção seguida de inserção). Note que o campo *Vagas* sempre terá 1 byte.

Arquivo de Índices

No cenário atual, **os índices não cabem em RAM** e, portanto, para simular essa situação, dois "*Arquivos de Índices*: `iprimary` e `iride`" 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 `'T'` (True) ou não `'F'` (False);
   - 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: `GABC`, `LCAC` e `AFGZ` (nesta ordem).

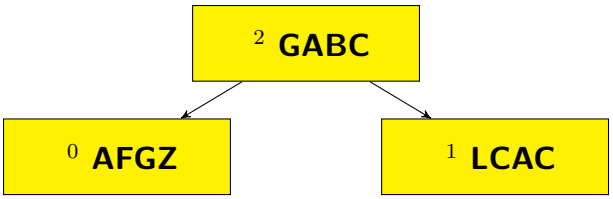
   Em disco, o arquivo de índice primário seria:

   ```
   001GABC2409070000##############T*********
   ```

   Em disco:

   ```
   002GABC2409070000LCAC1210070001T*********
   ```

   Note que ao inserir a chave iniciada por `AFGZ`, ocorre um *overflow* na raiz, 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 raiz. Visualmente falando:

   ```
   001AFGZ2409070002##############T*********
   001LCAC1210070001##############T*********
   001GABC2409070000##############F000001***
   ```

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

2. `iride`: índice secundário (Árvore-B), contendo suas chaves composta por: (*i*) a chave primária do respectivo registro e (*ii*) o destino concatenado com '`$`' a data (padrão `AAMMDD`) e a hora (`HHMM`) da carona, 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 + 41 bytes da *string*). Para as chaves não usadas, preencha todos os 51 bytes com '`#`'.
   - 1 byte para indicar se o nó é uma folha '`T`' (True) ou não '`F`' (False);
   - 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: '`SAO PAULO$1910301500`' (`SAO`), '`RIO DE JANEIRO$1910120700`' (`RIO`) e '`UFSCAR$1909240730`' (`UFS`).


```
003LCAC121007RIO DE JANEIRO$1910120700###############
#GABC240907SAO PAULO$1910301500####################A
FGZ240907UFSCAR$1909240730#############T***********
```

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 raiz de cada árvore. Assuma que um nó do índice corresponde a uma página e, portanto, cabe no *buffer* de memória. Dessa forma, trabalhe apenas com 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.

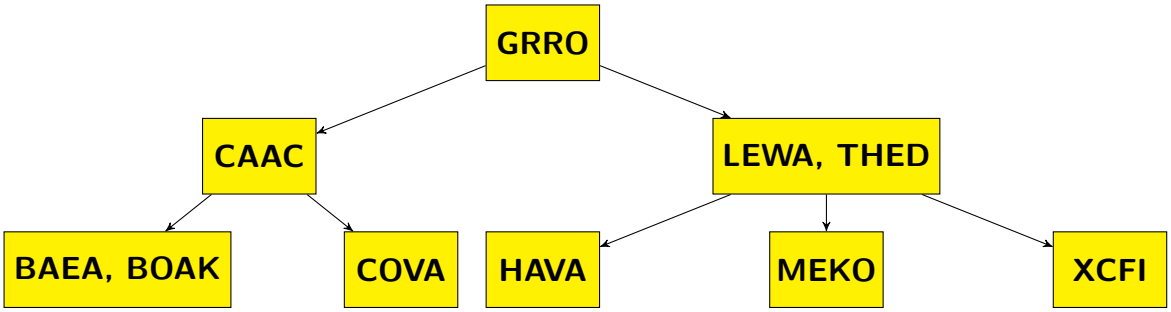
As seguintes operações devem ser fornecidas (nessa ordem):

1. **Cadastro.** O usuário deve ser capaz de inserir uma nova carona. Seu programa deve ler os seguintes campos (nessa ordem): **nome do motorista**, **gênero**, **data de nascimento**, **celular**, **modelo do veículo**, **placa**, **trajeto**, **data da carona**, **hora**, **valor** e **quantidade de vagas**. 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, exceto na opção de **Alteração**. 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 a quantidade de vagas de uma carona informando a sua chave primária. Caso ela 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 (1 byte com o valor entre 0 e 9) 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 uma carona:
   - **1. por código:** solicitar ao usuário a chave primária. Caso a carona não exista, seu programa deve exibir a mensagem "`Registro não encontrado!\n`" e retornar ao menu principal. Caso a carona exista, todos os seus dados devem 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: `LEWA041200`, `MEKO140118`, `BOAK241103`, `THED271000`, `HAVA160314`, `CAAC180614`, `COVA011221`, `GRRO120803`, `BAEA240318` e `XCFI201105`.



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

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

MEKO140118
MARIO
M
24/09/1999
(15) 99815-1234
BELINA
EKO-1234
AFONSO
VERGUEIRO|PANNUNZIO|UFSCAR
14/01/20
18:30
004.50
3
**********BUSCAR**********
Busca por CAAC999999. Nos percorridos:
GRRO120803
CAAC180614
COVA011221

Registro(s) nao encontrado!
```

- **2. por destino, data e hora:** solicitar ao usuário o destino seguido do data (formato `DD/MM/AA`) e hora (formato `HH:MM`). Caso nenhuma carona tenha sido encontrada, o programa deve exibir a mensagem "`Registro(s) nao encontrado!`" e retornar ao menu

principal. Exemplo de utilização para as buscas: {'UFSCAR','14/01/20','18:30'} e {'SAO PAULO','10/12/19','10:00'}

```
**********BUSCAR**********
Busca por UFSCAR$2001141830.
Nos percorridos:
AMERICANA$1910191030, MARILIA$1910012000, UBERLANDIA$2001051200
UFSCAR$2001141830, VOTORANTIM$1912280600

MEKO140118
MARIO
M
24/09/1999
(15) 99815-1234
BELINA
EKO-1234
AFONSO
VERGUEIRO|PANNUNZIO|UFSCAR
14/01/20
18:30
004.50
3
**********BUSCAR**********
Busca por SAO PAULO$1912101000.
Nos percorridos:
AMERICANA$1910191030, MARILIA$1910012000, UBERLANDIA$2001051200

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**. Caso não haja nenhum registro, imprima a mensagem de 'Registro(s) nao encontrado!' 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. Por exemplo, considere a árvore-B apresentada acima, a sua listagem resultaria em:

```
**********LISTAR**********
1 - GRRO120803
2 - CAAC180614
3 - BAEA240318, BOAK241103
3 - COVA011221
2 - LEWA041200, THED271000
```

```
3 - HAVA160314
3 - MEKO140118
3 - XCFI201105
```

- **2. Árvore-B secundária:** realiza uma travessia **em ordem**, exibindo todos as caronas na ordem lexicográfica do destino e data/hora. Exemplo:

```
**********LISTAR**********
AMERICANA---------- 19/10/19 - 10:30-
AVARE-------------- 22/11/19 - 22:00-
MARILIA------------ 30/10/19 - 20:00-
RIO DE JANEIRO----- 05/01/20 - 08:15-
SAO PAULO---------- 12/12/19 - 15:40-
SOROCABA----------- 10/10/19 - 18:00-
SOROCABA----------- 09/11/19 - 07:50-
UBERLANDIA--------- 05/01/20 - 12:00-
UFSCAR------------- 14/01/20 - 07:00-
UFSCAR------------- 14/01/20 - 07:30-
UFSCAR------------- 14/01/20 - 08:10-
```

Note que cada linha é composta por 49 bytes: 30 caracteres (destino) + 1 espaço em branco + 8 caracteres (data) + 1 espaço em branco + 1 caractere (traço) + 1 espaço em branco + 5 caracteres (hora) + 1 caractere (traço) + 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 pronto 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 sua produtividade.

CUIDADOS

1. O projeto deverá ser submetido no `Judge` em um único arquivo com o nome `{RA}_ORI_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.