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

Trabalho 02
Árvore-B

Prazo para entrega: 01/03/2023 – 23:59

Atenção

- **Sistema de submissão:** http://judge.sor.ufscar.br/ori/
- **Arquivo:** deverá ser submetido um único código-fonte seguindo o padrão de nomenclatura `<RA>_ORI_T02.c`, ex: `123456_ORI_T02.c`;
- **E/S:** tanto a entrada quanto a saída de dados devem ser de acordo com os casos de testes abertos;
- **Identificadores de variáveis:** escolha nomes apropriados;
- **Documentação:** inclua comentários e indente corretamente o programa;
- **Erros de compilação:** nota **zero** no trabalho;
- **Tentativa de fraude:** nota **zero na média** para todos os envolvidos. Fraudes, como tentativas de compras de soluções ou cópias de parte ou de todo código-fonte, de qualquer origem, implicará na reprovação direta na disciplina. Partes do código cujas **ideias** foram desenvolvidas em colaboração com outro(s) aluno(s) devem ser devidamente documentadas em comentários no referido trecho. Contudo, isso **NÃO** autoriza a cópia de trechos de código, a codificação em conjunto, compra de soluções, ou compartilhamento de tela para resolução do trabalho. Em resumo, você pode compartilhar ideias em alto nível, modos de resolver o problema, mas não o código;
- Utilize o código-base e as mensagens pré-definidas (`#define`).

1 Contexto

Sua plataforma de distribuição de cursos online, a **UFSCourses**, se tornou um grande sucesso. Agora, milhares de usuários passaram a usá-la diariamente para adquirir tanto cursos nacionais como internacionais. Consequentemente, a quantidade de usuários, cursos e inscrições têm crescido abruptamente.

Acompanhando métricas de qualidade do seu sistema, como quantidade de requisições e tempo de resposta das operações, além das métricas do banco de dados, percebeu-se que algumas operações começaram a ficar lentas. Cadastros de usuários e cursos são rápidos, mas a manutenção dos índices tem ficado demasiadamente lenta; buscas por inscrições demoram; listagens estão demandando muito tempo, entre outros problemas de lentidão do sistema. Por meio de uma análise, foi constatado que o gargalo deve-se ao fato de índices simples (listas ordenadas) não serem mais adequados, pois não cabem mais na memória RAM, demandando muitos acessos ao disco para realizar as operações.

Após uma avaliação técnica, percebeu-se que a estrutura mais adequada para se utilizar no armazenamento de índices de grande volume de dados é a *Árvore-B*.

2 Base de dados da aplicação

O sistema será composto por dados dos usuários, dos cursos e das inscrições, conforme descrito a seguir.

2.1 Dados dos usuários

- **id_usuario**: identificador único de um usuário (chave primária), composto por 11 números. Não poderá existir outro valor idêntico na base de dados. Ex: `57956238064`;
- **nome**: nome do usuário. Ex: `Mefistofelio Credolfo`;
- **email**: e-mail do usuário. Ex: `mefis.credo@mail.com`;
- **telefone**: número no formato `<99><999999999>`. Ex: `15924835754`;
- **saldo**: saldo que o usuário possui na sua conta, no formato `<9999999999>.<99>`. Ex: `0000004605.10`;

2.2 Dados dos cursos

- **id_curso**: identificador numérico único para cada curso, composto por 8 dígitos. Ex: `01234567`;
- **titulo**: título do curso;
- **instituicao**: nome da instituição responsável pelo curso;
- **ministrante**: nome do ministrante responsável pelo curso;
- **lancamento**: data em que o curso foi lançado, no formato `<AAAA><MM><DD>`. Ex: `20170224`;

- **carga**: carga horaria do curso (em horas), no formato `<9999>`. Ex: `0060`;
- **valor**: valor do curso no formato `<999999999999>.<99>`. Ex: `00000000027.99`
- **categorias**: até três categorias relacionadas com o conteúdo do curso, separadas por um '|'. Ex: `Algoritmos|Estruturas de Dados|Python`

2.3 Dados das inscrições

- **id_curso**: ID do curso em que o usuário está se inscrevendo;
- **id_usuario**: ID do usuário que está realizando a inscrição no curso;
- **data_inscricao**: data em que o usuário realizou a inscrição no curso, no formato `<AAAA><MM><DD><HH><MM>`. Ex: `202201021020`;
- **status**: indica se a inscrição está Ativa (`A`), Inativa (`I`) ou se o usuário já concluiu o curso (`C`).
- **data_atualizacao**: data da última atualização do status da inscrição, no formato `<AAAA><MM><DD><HH><MM>`. Ex: `202201021045`;

Garantidamente, nenhum campo de texto receberá caracteres acentuados.

2.4 Criação das bases de dados em SQL

Cada base de dados corresponde a um arquivo distinto. Elas poderiam ser criadas em um banco de dados relacional usando os seguintes comandos SQL:

```
CREATE TABLE usuarios (
  id_usuario  varchar(11) NOT NULL PRIMARY KEY,
  nome        text NOT NULL,
  email       text NOT NULL,
  telefone    varchar(11) NOT NULL,
  saldo       numeric(12, 2) DEFAULT 0,
);
CREATE TABLE cursos (
  id_curso     varchar(8) NOT NULL PRIMARY KEY,
  titulo       text NOT NULL,
  instituicao  text NOT NULL,
  ministrante  text NOT NULL,
  lancamento   varchar(8) NOT NULL,
  carga        numeric(4, 0) NOT NULL DEFAULT 0,
  valor        numeric(12, 2) NOT NULL DEFAULT 0,
  categorias   text[3] DEFAULT '{}'
);
```

```
CREATE TABLE inscricoes (
  id_curso          varchar(8) NOT NULL,
  id_usuario        varchar(11) NOT NULL,
  data_inscricao    varchar(12) NOT NULL,
  status            varchar(1) NOT NULL,
  data_atualizacao  varchar(12) NOT NULL,
);
```

3 Operações suportadas pelo programa

Os dados devem ser manipulados através do console/terminal (modo texto) usando uma sintaxe similar à SQL, sendo que as operações a seguir devem ser fornecidas.

3.1 Cadastro de usuários

```
INSERT INTO usuarios VALUES ('<id_usuario>', '<nome>', '<email>', '<telefone>');
```

Para criar uma nova conta de usuário, seu programa deve ler os campos `id_usuario`, `nome`, `email` e `telefone`. Inicialmente, a conta será criada sem saldo (00000000000.00). Caso o número do telefone não seja informado, este campo deverá ser preenchido com '*' (asterisco) para todos os dígitos. Todos os campos fornecidos serão dados de maneira regular, não havendo a necessidade de qualquer pré-processamento da entrada. A função deve falhar caso haja a tentativa de inserir um usuário com ID já cadastrado, ou seja, com ID que já esteja no sistema. Neste caso, deverá ser apresentada a mensagem de erro padrão `ERRO_PK_REPETIDA`. Caso a operação se concretize com sucesso, exibir a mensagem padrão `SUCESSO`.

Lembre-se de atualizar todos os índices necessários durante a inserção.

3.2 Atualização de telefone

```
UPDATE usuarios SET telefone = '<telefone>' WHERE id_usuario = '<id_usuario>';
```

O usuário deverá poder atualizar seu número de telefone dado o ID do usuário da conta e o número novo. Caso o usuário não esteja cadastrado no sistema, o programa deve imprimir a mensagem `ERRO_REGISTRO_NAO_ENCONTRADO`. Senão, o programa deve atualizar o telefone do usuário e imprimir a mensagem padrão `SUCESSO`.

3.3 Adicionar saldo na conta

```
UPDATE usuarios SET saldo = saldo + '<valor>' WHERE id_usuario = '<id_usuario>';
```

O usuário deverá ser capaz de adicionar valor na sua conta dado seu ID e o valor desejado. Caso o usuário não esteja cadastrado no sistema, o programa deve imprimir a mensagem padrão `ERRO_REGISTRO_NAO_ENCONTRADO`. Caso o valor que esteja sendo adicionado seja menor ou igual a

zero, o programa deve imprimir a mensagem ERRO_VALOR_INVALIDO. Se não houver nenhum desses problemas, o saldo deverá ser atualizado, seguido da impressão da mensagem padrão de SUCESSO.

3.4 Cadastro de cursos

```
INSERT INTO cursos VALUES ('<titulo>', '<instituicao>', '<ministrante>', '<lanca-
mento>', '<carga>', '<valor>');
```

Para um curso ser adicionado no banco de dados, seu programa deverá ler os campos que contém o título, instituicao, ministrante, lançamento, carga e valor. Assim como no cadastro de usuário, todos os campos serão fornecidos de maneira regular, não havendo a necessidade de pré-processamento. O ID do curso (id_curso) deverá ser preenchido de acordo com a quantidade de cursos cadastrados no sistema, ou seja, é um valor incremental. A função deve falhar caso haja a tentativa de inserir um curso cujo título já esteja presente no banco de dados, e deverá ser apresentada a mensagem de erro padrão ERRO_PK_REPETIDA. Caso a operação se concretize, exiba a mensagem padrão SUCESSO.

3.5 Atribuir categoria a um curso

```
UPDATE cursos SET categorias = array_append(categorias, '<categorias>') WHERE
titulo = '<titulo do curso>';
```

O usuário deve poder adicionar uma categoria a um ou mais cursos dado seu título e a categoria escolhida. Caso o curso não exista, o programa deve imprimir ERRO_REGISTRO_NAO_ENCONTRADO e, caso a categoria nova já esteja presente nas categorias do curso, seu programa deve imprimir ERRO_CATEGORIA_REPETIDA. Existe um máximo de três categorias por curso e, garantidamente, não haverá nenhuma tentativa de inserir mais de três por curso. Caso não haja nenhum erro, o programa deve atribuir a categoria ao curso, atualizando todos os índices e arquivos necessários e, então, imprimir a mensagem padrão SUCESSO.

3.6 Inscrições em cursos

```
INSERT INTO inscricoes VALUES ('<id_curso>', '<id_usuario>');
```

O usuário poderá inscrever-se em um curso dado o ID do curso e o ID do usuário. Caso o curso ou o usuário não existam, o programa deve imprimir ERRO_REGISTRO_NAO_ENCONTRADO. Caso a inscrição já tenha sido realizada, o seu programa deve imprimir ERRO_PK_REPETIDA. Primeiro, é preciso checar se o curso e o usuário existem. Caso o saldo presente na conta do usuário seja insuficiente para realizar a inscrição, ela não deve ser efetuada e a mensagem padrão ERRO_SALDO_NAO_SUFICIENTE deve ser impressa. Caso não haja nenhum desses problemas, a inscrição deverá ser realizada, gravando a data em que a foi feita, setando o status da inscrição para (A)tiva, gravando a data em que o status da inscrição foi atualizado, atualizando os índices necessários e imprimindo a mensagem padrão SUCESSO.

3.7 Remoção de usuários

```
DELETE FROM usuarios WHERE id_usuario = '<id_usuario>';
```

O usuário deverá ser capaz de remover uma conta dado um ID de usuário. Caso a conta não exista, seu programa deverá exibir a mensagem padrão `ERRO_REGISTRO_NAO_ENCONTRADO`. A remoção na base de dados deverá ser feita por meio de um marcador, conforme descrito na Seção 5. Se a operação for realizada, exibir a mensagem padrão `SUCESSO`.

3.8 Atualização de status de inscrições

```
UPDATE inscricoes SET status = '<status>' WHERE id_curso = (SELECT id_curso FROM cursos WHERE titulo = '<titulo do curso>') AND id_usuario = '<id_usuario>';
```

Atualiza o status da inscrição de um usuário em um ou mais cursos, dados o título do curso e o ID do usuário. Caso o usuário não esteja inscrito no curso, o programa deve imprimir `ERRO_REGISTRO_NAO_ENCONTRADO`. Caso contrário, o status da inscrição deverá ser atualizado, gravando a data em que o status foi atualizado e setando o novo status para (`A`)tiva, (`I`)nativa ou (`C`)oncluído. Por fim, imprimir a mensagem padrão `SUCESSO`.

3.9 Busca

As seguintes operações de busca por usuários e cursos deverão ser implementadas. *Em todas elas, será necessário utilizar a busca binária e mostrar o caminho percorrido nos índices.* Antes dos resultados das buscas, deverá ser impresso os RRNs dos nós percorridos, assim como o percurso feito dentro de cada nó. Exemplo:

```
Nos percorridos:   2 (1 0) 0 (1 0)
```

No exemplo acima, a árvore tem ordem $m = 3$. Fora dos parênteses estão os RRNs dos nós da árvore que foram percorridos e, dentro dos parênteses, o percurso de índices dentro de cada nó, que no caso poderia ser 0, 1 ou 2. Observe que, caso o número de elementos dentro do nó seja par (p.ex, 10 elementos), então há 2 (duas) possibilidades para a posição da mediana dos elementos (p.ex., 5a ou 6a posição se o total fosse 10). Neste caso, **sempre** escolha a posição mais à direita (p.ex., a posição 6 caso o total for 10).

As seguintes operações de busca por usuários e cursos deverão ser implementadas.

3.9.1 Usuários

O usuário deverá poder buscar contas de usuários pelos seguintes atributos:

(a) ID do usuário:

```
SELECT * FROM usuarios WHERE id_usuario = '<id_usuario>';
```

Solicitar ao usuário o ID vinculado ao cadastro. Caso a conta não exista, seu programa deverá exibir a mensagem padrão `ERRO_REGISTRO_NAO_ENCONTRADO`. Caso a conta exista, todos os seus dados deverão ser impressos na tela de forma formatada.

3.9.2 Cursos

O usuário deverá ser capaz de buscar cursos pelos seguintes atributos:

(a) ID do curso:

```
SELECT * FROM cursos WHERE id_curso = '<id_curso>';
```

Solicitar ao usuário o ID do curso. Caso ele não exista no banco de dados, seu programa deverá exibir a mensagem padrão `ERRO_REGISTRO_NAO_ENCONTRADO`. Caso contrário, todos os dados do curso deverão ser impressos na tela de forma formatada.

(b) Pelo título do curso:

```
SELECT * FROM cursos WHERE titulo = '<titulo do curso>';
```

Caso o curso não seja encontrado, seu programa deverá exibir a mensagem padrão `ERRO_REGISTRO_NAO_ENCONTRADO`. Caso contrário, todos os dados deverão ser impressos na tela de forma formatada. Observe que, neste caso, como a busca será realizada em dois índices, será necessário exibir o caminho da busca binária para ambos: primeiro para o índice secundário e depois para o índice primário.

3.10 Listagem

As seguintes operações de listagem deverão ser implementadas.

3.10.1 Usuários

(a) Por IDs dos usuários:

```
SELECT * FROM usuarios ORDER BY id_usuario ASC;
```

Exibe todos os usuários ordenados de forma crescente pelo ID. Caso nenhum registro seja retornado, seu programa deverá exibir a mensagem padrão `AVISO_NENHUM_REGISTRO_ENCONTRADO`.

3.10.2 Cursos

(a) Por categoria:

```
SELECT * FROM cursos WHERE '<categoria>'= ANY (categorias) ORDER BY id_curso ASC
```

Exibe todos os cursos de uma determinada categoria, em ordem crescente de ID. Caso nenhum registro for retornado, seu programa deverá exibir a mensagem padrão `AVISO_NENHUM_REGISTRO_ENCONTRADO`.

Antes de listar os cursos, o seu programa deverá imprimir a sequência em que os registros foram percorridos tanto no índice primário quanto no secundário para a listagem (lembrando que a busca no índice secundário, que contém as categorias, deverá ser uma busca binária). Exemplo:

```
Registros percorridos:   2 1 0
Registros percorridos:   3 5 7
```

3.10.3 Inscrições

(a) Por data de inscrição:

```
SELECT * FROM inscricoes WHERE data_inscricao BETWEEN '<data_inicio>' AND
'<data_fim>' ORDER BY data_inscricao ASC;
```

Exibe todas as inscrições realizadas em um determinado período de tempo (`data` entre `<data_inicio>` e `<data_fim>`), em ordem cronológica. Ambas as datas estarão no formato `<AAAAMMDDHHMM>`. Para cada registro encontrado na listagem, deverá ser impresso o caminho percorrido. Caso nenhum registro for retornado, seu programa deverá exibir a mensagem padrão `AVISO_NENHUM_REGISTRO_ENCONTRADO`.

ATENÇÃO: antes de imprimir a lista de inscrições realizadas no período, primeiro é necessário imprimir o caminho percorrido para encontrar o registro cuja `data_inscricao` seja igual à `<data_inicio>` informada pelo usuário ou data posterior mais próxima.

3.11 Liberar espaço

```
VACUUM usuarios;
```

O arquivo de dados `ARQUIVO_USUARIOS` deverá ser reorganizado com a remoção física de todos os registros marcados como excluídos e os índices deverão ser atualizados. A ordem dos registros no arquivo "limpo" não deverá ser diferente do arquivo "sujo". Se a operação se concretizar, exibir a mensagem padrão `SUCESSO`.

3.12 Imprimir arquivos de dados

O sistema deverá imprimir os arquivos de dados da seguinte maneira:

(a) Dados dos usuários:

```
\echo file ARQUIVO_USUARIOS
```

Imprime o arquivo de dados de usuários. Caso esteja vazio, apresentar a mensagem padrão `ERRO_ARQUIVO_VAZIO`;

(b) Dados dos cursos:

`\echo file ARQUIVO_CURSOS`

Imprime o arquivo de dados de cursos. Caso esteja vazio, apresentar a mensagem padrão `ERRO_ARQUIVO_VAZIO`.

(c) Dados das inscrições:

`\echo file ARQUIVO_INSCRICOES`

Imprime o arquivo de inscrições feitas. Caso o arquivo esteja vazio, apresentar a mensagem padrão `ERRO_ARQUIVO_VAZIO`.

3.13 Imprimir índices primários

O sistema deverá imprimir os índices primários da seguinte maneira:

(a) Índice de usuários:
`\echo index usuarios_idx`

Imprime o arquivo de índice primário para usuários (`usuarios_idx`). Caso o arquivo esteja vazio, imprimir `ERRO_ARQUIVO_VAZIO`;

(b) Índice de cursos:
`\echo index cursos_idx`

Imprime o arquivo de índice primário para cursos (`cursos_idx`). Caso o arquivo esteja vazio, imprimir `ERRO_ARQUIVO_VAZIO`;

(c) Índice de inscrições:
`\echo index inscricoes_idx`

Imprime o arquivo de índice primário para compras (`inscricoes_idx`). Caso o arquivo esteja vazio, imprimir `ERRO_ARQUIVO_VAZIO`;

3.14 Imprimir índices secundários

O sistema deverá imprimir os índices secundários da seguinte maneira:

(a) Índice de título:
`\echo index titulo_idx`

Imprime o arquivo de índice secundário com o título dos cursos e os respectivos `id_cursos`. Caso o arquivo esteja vazio, imprimir `ERRO_ARQUIVO_VAZIO`.

(b) Índice de datas de inscrições com `data`, `id_curso` e `id_usuario`:

`\echo index data_curso_usuario_idx`

Imprime o índice secundário de datas das inscrições. Caso o índice esteja vazio, imprimir `ERRO_ARQUIVO_VAZIO`.

(c) Índice de categorias primário:
`\echo index categorias_primario_idx`

Imprime o índice secundário com o ID do curso em uma categoria (`categorias_primario_idx`) e o número de índice para o próximo curso da mesma categoria. Caso o índice esteja vazio, imprimir `ERRO_ARQUIVO_VAZIO`.

(d) Índice de categoria secundário:
`\echo index categorias_secundario_idx`

Imprime o índice secundário com as categorias dos cursos (`categorias_secundario_idx`) e o número de índice para o primeiro curso da categoria. Caso o índice esteja vazio, imprimir `ERRO_ARQUIVO_VAZIO`.

3.15 Finalizar

`\q`
Libera memória eventualmente alocada e encerra a execução do programa.

4 Arquivos de dados

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.

Deve-se utilizar as variáveis globais `ARQUIVO_USUARIOS`, `ARQUIVO_CURSOS` e `ARQUIVO_INSCRICOES` e as funções de leitura e escrita em *strings*, como `sprintf` e `sscanf`, para simular as operações de leitura e escrita em arquivo. Os arquivos de dados devem ser no formato ASCII (arquivo texto).

`ARQUIVO_USUARIOS`: deverá ser organizado em registro de tamanho fixo de 128 *bytes* (128 caracteres). Os campos `nome` (tamanho máximo de 44 *bytes*) e `email` (tamanho máximo de 44 *bytes*) devem ser de tamanho variável. Os demais campos devem ser de tamanho fixo, sendo eles `id_usuario` (11 *bytes*), `telefone` (11 *bytes*) e `saldo` (13 *bytes*). Portanto, os campos de tamanho fixo de um registro ocuparão 35 *bytes*. Os campos devem ser separados pelo caractere delimitador '`;`' (ponto e vírgula), e cada registro terá 5 delimitadores (um para cada campo). Caso o registro tenha menos de 128 *bytes*, o espaço restante deverá ser preenchido com o caractere '`#`'. Como são 35 *bytes* fixos + 5 *bytes* de

delimitadores, então os campos variáveis devem ocupar no máximo 88 *bytes*, para que o registro não exceda os 128 *bytes*.

Exemplo de arquivo de dados de usuários:

```
12345678910;Maria Eugenia Souza;mariaeugenia@mail.com;**********;
0000000030.00;##################################################9234
5678915;Joao da Silva;silva.joao@mail.com;**********;0000001999.8
7;##########################################################09898989
999;Manoel de Souza;manoel_souza@mail-server.com;15123451234;00000
00909.15;##################################################1111111111;
Clarita Leite Moca;leite.moca11@mymail.com.br;11999990000;00000000
00.00;##################################################10111213141;Mefi
stofelio Credolfo de Assis;credo-mefis@server.com.au;19987654321;0
00000987.65;################################
```

`ARQUIVO_CURSOS`: o arquivo de cursos deverá ser organizado em registros de tamanho fixo de 256 *bytes* (256 caracteres). Os campos `titulo` (máximo de 51 *bytes*), `instituicao` (máximo de 51 *bytes*), `ministrante` (máximo de 50 *bytes*) e `categoria` (máximo de 63 *bytes*, com no máximo 3 valores, onde cada categoria pode ter no máximo 20 *bytes*) devem ser de tamanhos variáveis. O campo multi-valorado `categoria` deve ter os seus valores separados por '`|`'. Os demais campos são de tamanho fixo e possuem as seguintes especificações: `id_curso` (8 *bytes*), `lancamento` (8 *bytes*), `carga` (4 *bytes*) e `valor` (13 *bytes*), totalizando 33 *bytes* de tamanho fixo. Assim como no registro de usuários, os campos devem ser separados pelo delimitador '`;`', cada registro terá 8 delimitadores para os campos e mais 3 para o campo multivalorado e, caso o registro tenha menos que 256 *bytes*, o espaço restante deverá ser preenchido com o caractere '`#`'. Como são 33 *bytes* de tamanho fixo + 8 *bytes* para os delimitadores, os campos variáveis devem ocupar no máximo 215 *bytes* para que não se exceda o tamanho do registro.

Exemplo de arquivo de dados de cursos:

```
00000000;Python for dummies;Stanford;George Washington;20221208;
0060;0000000099.99;Python|Computer Science|Programming|;########
################################################################
################################################################
00000001;Machine Learning;Harvard;Isaac Newton;20221210;0090;000
0000999.00;AI|Computer Science|Machine Learning|;###############
################################################################
################################################################
00000002;Data Structures;MIT;Thomas Cormen;20150208;0120;0000000
000.99;Data Structure|Computer Science|Programming|;############
################################################################
################################################################
00000003;Banco de Dados;UFSCar;Sahudy Gonzales;20201231;0060;000
0000000.00;Databases|Computer Science|Programming|;#############
################################################################
################################################################
```

`ARQUIVO_INSCRICOES`: o arquivo de inscricoes deverá ser organizado em registros de tamanho fixo de 44 *bytes*. Todos os campos possuem um tamanho fixo e, portanto, não é necessário a presença de delimitadores: `id_curso` (8 *bytes*), `id_usuario` (11 *bytes*), `data_inscricao` (12 *bytes*), `status` (1 *byte*) e `data_atualizacao` (12 *bytes*). No exemplo abaixo, os campos estão ordenados por `id_curso`, `data_inscricao`, `id_usuario`, `status` e `data_atualizacao`.

Exemplo de arquivo de dados de inscrições:

```
00000000 44565434213 202209201000 A 202209201000
00000004 37567876542 201909241355 I 202001102050
00000008 37567876542 202110030748 C 202212312359
00000005 66545678765 202210051617 A 202210051617
```

No exemplo acima, foram inseridos espaços em branco entre os campos apenas para maior clareza do exemplo. Note também, que não há quebras de linhas nos arquivos (elas foram inseridas apenas para facilitar a visualização da sequência de registros).

5 Instruções para as operações com os registros

- **Inserção**: cada usuário, curso e inscrição devem ser inseridos no final de seus respectivos arquivos de dados, e atualizados os índices.
- **Remoção**: o registro de um dado usuário deverá ser localizado acessando o índice primário (`id_usuario`). A remoção deverá colocar o marcador `*|` nas duas primeiras posições do registro removido. O espaço do registro removido não deverá ser reutilizado para novas inserções. Observe que o registro deverá continuar ocupando exatamente 128 *bytes*.

- **Atualização**: existem quatro campos alteráveis: (*i*) o telefone do usuário; (*ii*) o saldo do usuário; (*iii*) o status da inscrição; e (*iv*) a data da última atualização do status. Todos eles possuem tamanhos fixos e pré-determinados. Esses dados devem ser atualizados diretamente no registro, exatamente na mesma posição em que estiverem (em hipótese alguma o registro deverá ser removido e em seguida inserido).

6 Índices

No cenário atual, há um grande volume de dados, e nem mesmo os índices cabem em RAM. Para simular essa situação, a única informação que você deverá manter todo tempo em memória é o RRN da raiz de cada Árvore-B. 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. **É terminantemente proibido manter uma cópia dos índices inteiros em Tipos Abstratos de Dados (TADs) que não sejam Árvores-B ou, no caso das categorias, uma lista invertida**.

Todas as árvores terão a mesma ordem (m).

Durante a operação de inserção, caso ocorra *overflow*, **a chave promovida deverá ser sempre a maior chave da página à esquerda** do processo de divisão. Nesse caso, se a ordem (m) for par, a página à esquerda ficará com 1 (uma) chave a mais do que à direita no final do processo de divisão. Todo novo nó criado deverá ser inserido no final do respectivo arquivo de índice.

Durante a operação de remoção, caso:

1. **a chave a ser removida não esteja em um nó folha**, então, ela deverá primeiro ser substituída pela **maior chave da subárvore esquerda, ou seja, seu predecessor**;

2. ocorra *underflow*, será necessário emprestar chaves de um de seus nós irmãos (páginas à esquerda ou à direita), sempre que possível, dando **preferência para o nó irmão à direita**, exceto se o número de chaves nesse nó já for o mínimo ou se não houver um nó irmão à direita:

    - Se a redistribuição de chaves com um dos nós irmãos for possível, então **somente 1 (uma) chave deverá ser transferida** para o nó onde ocorreu o *underflow*;
    - Caso contrário, será necessário concatenar as chaves do nó onde ocorreu o *underflow* e de um de seus nós irmãos (páginas à esquerda ou à direita), sempre que possível, dando **preferência para o nó irmão à direita**, se houver. Nesse caso, a página à esquerda do processo de fusão receberá as chaves da página à direita, deixando-a vazia. O espaço ocupado por páginas vazias **não deverá ser reutilizado** em novas inserções.

A lista invertida de categorias também cresceu e, agora, será armazenada em um arquivo.

6.1 Índices primários

6.1.1 `usuarios_idx`

Índice primário do tipo Árvore-B que contém o `id_usuario` do usuário (chave primária) e o RRN do respectivo registro no arquivo de dados, ordenado pela chave primária.

Cada registro da Árvore-B para o índice `usuarios_idx` é composto por:

- *Contador de chaves:* 3 bytes para a quantidade de chaves;
- *Chaves ordenadas:* $(m-1)$ * (11 bytes de chave primária + 4 bytes para o RRN do arquivo de dados) bytes para armazenar as chaves. Para as chaves não usadas, preencha todos os bytes com '`#`';
- *Indicador de folha:* 1 byte para indicar se o nó é folha '`T`' (`True`) ou não '`F`' (`False`);
- *Apontadores para os filhos:* (m * 3 bytes para cada RRN filho) 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.

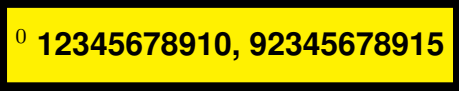
Exemplo da representação da Árvore-B de ordem 3, após a inserção das chaves: **12345678910**, **92345678915**, **09898989999**, **11111111111** e **10111213141** (nesta ordem).

- Inserindo a chave `12345678910`:

Em disco, o arquivo de índice primário seria:

```
001 | 12345678910 0000 | ########## #### | T | *** *** ***
```

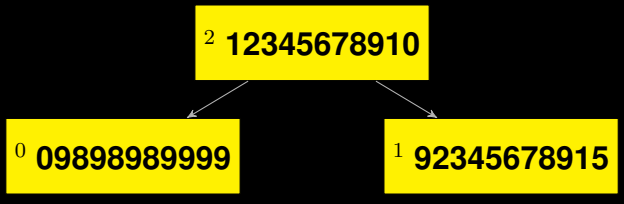
- Inserindo a chave `92345678915`:



Em disco:

```
002 | 12345678910 0000 | 92345678915 0001 | T | *** *** ***
```

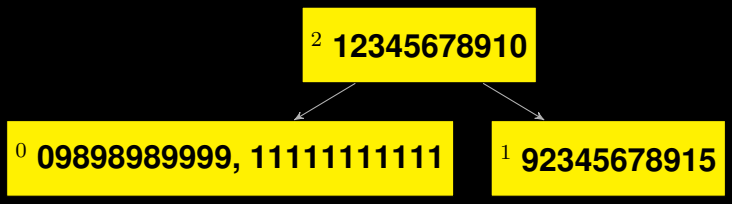
- Inserindo a chave `09898989999`:



Ao inserir a chave iniciada por 09898989999, 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. Portanto:

```
001 | 09898989999 0002 | ########## #### | T | *** *** ***
001 | 92345678915 0001 | ########## #### | T | *** *** ***
001 | 12345678910 0000 | ########## #### | F | 000 001 ***
```

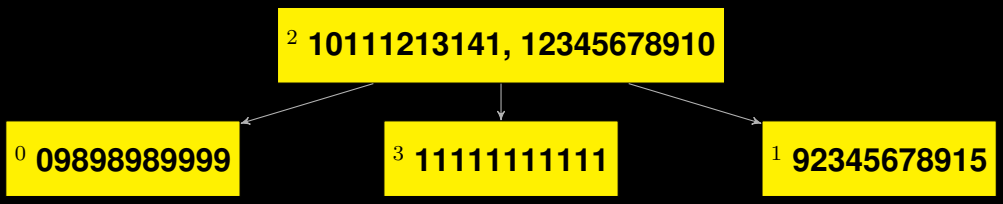
- Inserindo a chave 11111111111:



Em disco:

```
002 | 09898989999 0002 | 11111111111 0003 | T | *** *** ***
001 | 92345678915 0001 | ########## #### | T | *** *** ***
001 | 12345678910 0000 | ########## #### | F | 000 001 ***
```

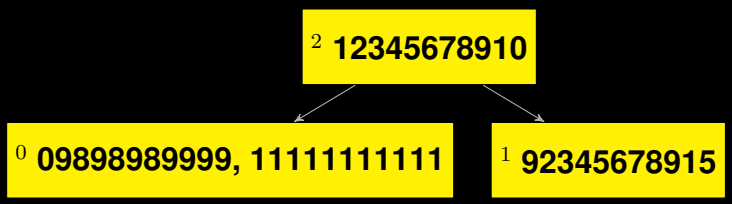
- Inserindo a chave 10111213141:



Ao inserir a chave iniciada por 10111213141, ocorre um *overflow* no nó de RRN 0 (página à esquerda do processo de divisão), sendo necessário criar 1 novo nó de RRN 3 (página à direita do processo de divisão) para a redistribuição de chaves e promover a chave 10111213141 para o nó raiz. Portanto:

```
001 | 09898989999 0002 | ########## #### | T | *** *** ***
001 | 92345678915 0001 | ########## #### | T | *** *** ***
002 | 10111213141 0004 | 12345678910 0000 | F | 000 003 001
001 | 11111111111 0003 | ########## #### | T | *** *** ***
```

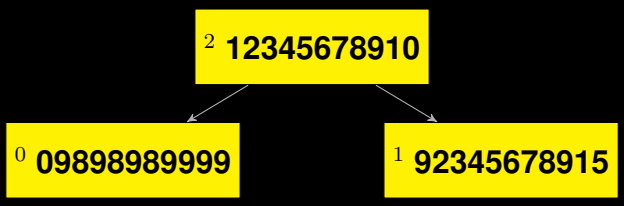
- Removendo a chave 10111213141:



Ao remover a chave iniciada por `10111213141`, como ela se encontra no nó raiz (de RRN 2) e esse nó não é folha, então, ela é primeiro substituída pela chave iniciada por `09898989999`, que é a sua predecessora, isto é, maior chave do nó de RRN 0 (subárvore esquerda). Em seguida, ela é removida do nó de RRN 0, causando um *underflow* nesse nó, sendo necessário emprestar chaves do nó de RRN 3 (página à direita). Como o número de chaves nesse nó já é o mínimo, então é necessário fundir ambos os nós (de RRN 0 e 3 respectivamente). O primeiro (página à esquerda), será usado para a concatenação de chaves, já o segundo (página à direita), ficará vazio. Isso fará com que a chave iniciada por `09898989999` seja rebaixada do nó raiz (de RRN 2). Portanto:

```
002 | 09898989999 0002 | 11111111111 0003 | T | *** *** ***
001 | 92345678915 0001 | ########### #### | T | *** *** ***
001 | 12345678910 0000 | ########### #### | F | 000 001 ***
000 | ########### #### | ########### #### | T | *** *** ***
```

- Removendo a chave `11111111111`:



Em disco:

```
001 | 09898989999 0002 | ########### #### | T | *** *** ***
001 | 92345678915 0001 | ########### #### | T | *** *** ***
001 | 12345678910 0000 | ########### #### | F | 000 001 ***
000 | ########### #### | ########### #### | T | *** *** ***
```

- Removendo a chave `09898989999`:

Ao remover a chave iniciada por `09898989999`, ocorre um *underflow* no nó de RRN 0, sendo necessário emprestar chaves do nó de RRN 1 (página à direita). Como o número de chaves nesse nó já é o mínimo, então, é necessário fundir ambos os nós (de RRN 0 e 1, respectivamente). O primeiro (página à esquerda), será usado para a concatenação de chaves, já o segundo (página à direita), ficará vazio. Isso fará com que a chave iniciada por `12345678910` seja rebaixada do nó raiz (de RRN 2), deixando-o vazio. Por esse motivo, o nó de RRN 0 é definido como sendo o novo nó raiz. Portanto:

```
002 | 12345678910 0000 | 92345678915 0001 | T | *** *** ***
000 | ########### #### | ########### #### | T | *** *** ***
000 | ########### #### | ########### #### | F | *** *** ***
000 | ########### #### | ########### #### | T | *** *** ***
```

- Removendo a chave `92345678915`:

Em disco:

```
001 | 12345678910 0000 | ########## #### | T | *** *** ***
000 | ########## #### | ########## #### | T | *** *** ***
000 | ########## #### | ########## #### | F | *** *** ***
000 | ########## #### | ########## #### | T | *** *** ***
```

- Removendo a chave `12345678910`, a árvore ficará vazia e a representação em disco será:

```
000 | ########## #### | ########## #### | T | *** *** ***
000 | ########## #### | ########## #### | T | *** *** ***
000 | ########## #### | ########## #### | F | *** *** ***
000 | ########## #### | ########## #### | T | *** *** ***
```

Em resumo, um nó da Árvore-B de `usuarios_idx`, de ordem m, possui as seguintes informações:

```
{QUANTIDADE DE CHAVES}
{ID_USUARIO_1} {RRN_1}
{ID_USUARIO_2} {RRN_2}
...
{ID_USUARIO_(m-1)} {RRN_(m-1)}
{FOLHA}
{RRN FILHO_1} {RRN FILHO_2} ... {RRN_FILHO_m}
```

Note que, não há quebras de linhas no arquivo, espaços ou *pipes* (‘`|`’), Eles foram inseridos apenas para exemplificar a sequência de registros.

6.1.2 `cursos_idx`

Índice primário do tipo Árvore-B que contém o `id_curso` do jogo (chave primária) e o RRN do respectivo registro no arquivo de dados, ordenado pela chave primária.

Cada registro da Árvore-B para o índice `cursos_idx` é composto por:

- *Contador de chaves:* 3 bytes para a quantidade de chaves;
- *Chaves ordenadas:* $(m-1)$ * (8 bytes de chave primária + 4 bytes para o RRN do arquivo de dados) bytes para armazenar as chaves. Para as chaves não usadas, preencha todos os bytes com ‘`#`’;
- *Indicador de folha:* 1 byte para indicar se o nó é folha ‘`T`’ (`True`) ou não ‘`F`’ (`False`);

- *Apontadores para os filhos:* (m * 3 bytes para cada RRN filho) 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: `43487689`, `53999519` e `39440633` (nesta ordem).

- Inserindo a chave `43487689`:

Em disco, o arquivo de índice primário seria:

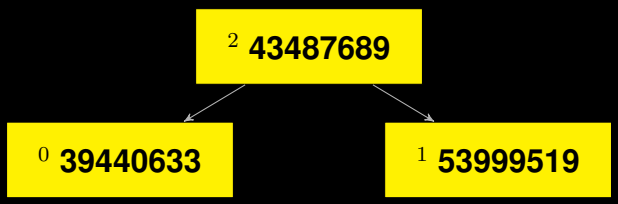
```
001 | 43487689 0000 | ######## #### | T | *** *** ***
```

- Inserindo a chave `53999519`:

Em disco:

```
002 | 43487689 0000 | 53999519 0001 | T | *** *** ***
```

- Inserindo a chave `39440633`:



Ao inserir a chave `39440633`, 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. Portanto:

```
001 | 39440633 0002 | ######## #### | T | *** *** ***
001 | 53999519 0001 | ######## #### | T | *** *** ***
001 | 43487689 0000 | ######## #### | F | 000 001 ***
```

Em resumo, um nó da Árvore-B de `cursos_idx`, de ordem m, possui as seguintes informações:

```
{QUANTIDADE DE CHAVES}
{ID_CURSO_1} {RRN_1}
{ID_CURSO_2} {RRN_2}
...
{ID_CURSO_(m-1)} {RRN_(m-1)}
{FOLHA}
{RRN FILHO_1} {RRN FILHO_2} ... {RRN_FILHO_m}
```

Note que, não há quebras de linhas no arquivo, espaços ou *pipes* ('|'), Eles foram inseridos apenas para exemplificar a sequência de registros.

6.1.3 `inscricoes_idx`

Índice primário que consiste no ID do curso em que o usuário se inscreveu, o ID do usuário que realizou a inscrição e o RRN relativo ao registro no arquivo de inscrições, ordenado pelo ID do curso (`id_curso`) e o ID do usuário (`id_usuario`).

Cada registro da Árvore-B para o índice `inscricoes_idx` é composto por:

- *Contador de chaves:* 3 bytes para a quantidade de chaves;
- *Chaves ordenadas:* $(m-1)$ * (19 bytes de chave primária [8 bytes para o `id_curso` + 11 bytes para o `id_usuario`] + 4 bytes para o RRN do arquivo de dados) bytes para armazenar as chaves. Para as chaves não usadas, preencha todos os bytes com '`#`';
- *Indicador de folha:* 1 byte para indicar se o nó é folha '`T`' (True) ou não '`F`' (False);
- *Apontadores para os filhos:* (m * 3 bytes para cada RRN filho) 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 de arquivo de índice primário de inscrições representado por uma Árvore-B de ordem 4, após a inserção das chaves na ordem: '`44678965 25091500437 0002`', '`34678965 20210912321 0000`' e '`51478965 20210101098 0001`'.


```
003 | 34678965 20210912321 0000 | 44678965 25091500437 0002 |
      51478965 20210101098 0001 | T | *** *** *** ***
```

Em resumo, um nó da Árvore-B de compras, de ordem m, possui as seguintes informações:

```
{QUANTIDADE DE CHAVES}
{ID_CURSO_1} {ID_USUARIO_1} {RRN_1}
{ID_CURSO_2} {ID_USUARIO_2} {RRN_2}
...
{ID_CURSO_(m-1)} {ID_USUARIO_(m-1)} {RRN_(m-1)}
{FOLHA}
{RRN FILHO_1} {RRN FILHO_2} ... {RRN_FILHO_m}
```

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

6.2 Índices secundários

6.2.1 `titulo_idx`

Índice do tipo Árvore-B que contém os cursos ordenados por títulos (em ordem lexicográfica) e a chave primária (`id_curso`) do curso específico.

Cada registro da Árvore-B para o índice `titulo_idx` é composto por:

- *Contador de chaves:* 3 bytes para a quantidade de chaves;
- *Chaves ordenadas:* $(m-1)$ * (51 bytes de título + 8 bytes de `id_curso`) bytes para armazenar as chaves. Para as chaves não usadas ou para completar o tamanho da chave de título, preencha todos os bytes com '`#`';
- *Indicador de folha:* 1 byte para indicar se o nó é folha '`T`' (True) ou não '`F`' (False);
- *Apontadores para os filhos:* (m * 3 bytes para cada RRN filho) bytes para indicar os RRNs dos nós descendentes do nó atual. 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 nula.

Exemplo de arquivo de índice secundário de título, representado por uma Árvore-B de ordem 5, após a inserção da chave '`Python for dummies 46578965`'.


```
001 | PYTHON FOR DUMMIES################################## 46578965 |
    ################################################### ######## |
    ################################################### ######## |
    ################################################### ######## |
    T | *** *** *** *** ***
```

Em resumo, um nó da Árvore-B de títulos de cursos, de ordem m, possui as seguintes informações:

```
{QUANTIDADE DE CHAVES}
{TITULO_1} {ID_CURSO_1}
{TITULO_2} {ID_CURSO_2}
...
{TITULO_(m-1)} {ID_CURSO_(m-1)}
{FOLHA}
{RRN FILHO_1} {RRN FILHO_2} ... {RRN_FILHO_m}
```

Lembre-se, aqui também não há quebras de linhas no arquivo, espaços ou *pipes* ('`|`'), Eles foram inseridos apenas para exemplificar a sequência de registros.

6.2.2 `data_curso_usuario_idx`

Índice secundário do tipo Árvore-B que contém as datas das inscrições (em ordem cronológica), o ID do curso (em ordem crescente) e o ID do usuário (em ordem crescente).

Cada registro da Árvore-B para o índice `data_curso_usuario_idx` é composto por:

- *Contador de chaves:* 3 bytes para a quantidade de chaves;
- *Chaves ordenadas:* $(m-1)$ * (12 bytes para a data de inscrição + 19 bytes das chaves primárias [8 bytes para o `id_curso`) + 11 bytes para o `id_usuario`)]) bytes para armazenar as chaves. Para as chaves não usadas, preencha todos os bytes com '`#`';

- *Indicador de folha:* 1 byte para indicar se o nó é folha '`T`' (True) ou não '`F`' (False);
- *Apontadores para os filhos:* (m * 3 bytes para cada RRN filho) bytes para indicar os RRNs dos nós descendentes do nó atual. 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 nula.

Exemplo de arquivo de índice secundário representado por uma Árvore-B de ordem 4, após a inserção das chaves na ordem: '`202103251000 00000001 44678965437`', '`202109121100 01234567 34678965321`' e '`202101011200 00000002 51478965098`'.


```
003 | 202101011200 00000002 51478965098 | 202103251000 00000001 44678965437 |
      202109121100 01234567 34678965321 | T | *** *** *** ***
```

Em resumo, um nó da Árvore-B de datas das inscrições, de ordem m, possui as seguintes informações:

```
{QUANTIDADE DE CHAVES}
{DATA_INSCRICAO_1} {ID_CURSO_1} {ID_USUARIO_1}
{DATA_INSCRICAO_2} {ID_CURSO_2} {ID_USUARIO_2}
...
{DATA_INSCRICAO_(m-1)} {ID_CURSO_(m-1)} {ID_USUARIO_(m-1)}
{FOLHA}
{RRN FILHO_1} {RRN FILHO_2} ... {RRN_FILHO_m}
```

Novamente, não há quebras de linhas no arquivo, espaços ou *pipes* ('`|`'), Eles foram inseridos apenas para exemplificar a sequência de registros.

6.2.3 `categorias_idx`

Índice secundário do tipo *lista invertida*. Será necessário manter dois índices (`categorias_primario_idx` e `categorias_secundario_idx`), sendo que o primário possui os ID de um curso (`id_curso`) da categoria e o apontador para o próximo curso da mesma categoria nesse mesmo índice primário. Se não houver um próximo curso, esse apontador deve possuir o valor -1. No índice secundário estão as categorias, assim como a referência do primeiro curso daquela categoria no índice primário. Para padronizar o índice secundário, complete a categoria com '`#`' para ocupar exatamente 20 bytes. Da mesma forma, preencha os apontadores para ocuparem 4 bytes e no caso do -1 represente como '`-001`'.

Para simplificar o entendimento, considere o seguinte exemplo:

Categorias primário	
ID do curso	próx. registro
0	4
4	-1
4	7
4	5
3	-1
7	6
5	-1
5	-1
...	...

Categorias secundário	
Categoria	registro
ALGORITMOS	0
APRENDIZADO DE MAQUINA	1
PYTHON	3
VISAO COMPUTACIONAL	2
...	...

No exemplo acima, a tabela de categorias secundário possui a categoria na primeira coluna, assim como o RRN do primeiro curso daquela categoria que foi inserido na tabela de categorias primário. Na tabela primária, tem-se na primeira coluna o ID dos cursos em cada categoria. Note que o curso com ID = 4 aparece três vezes no exemplo, o que significa que ele pertence a três categorias diferentes. Na segunda coluna da tabela primária, temos o RRN para o próximo curso de mesma categoria na própria tabela de categorias primária, sendo que, RNN = -1, significa que aquele curso é o último de sua categoria. Vale destacar que o índice primário **não** precisa estar organizado, pois cada registro já possui uma referência direta para o próximo (assim como em uma lista encadeada).

Exemplo de como as tabelas acima ficariam nos arquivos de índices primário e secundário:

```
Índice primário:
00000000 0004 00000004 -001 00000004 0007
00000004 0005 00000003 -001 00000007 0006
00000005 -001 00000005 -001


Índice secundário:
ALGORITMOS########## 0000 APRENDIZADO DE MAQUI 0001
PYTHON############## 0003 VISAO COMPUTACIONAL# 0002
```

Lembrando novamente que as quebras de linha e espaços foram apenas adicionados para facilitar a visualização.

Deverá ser desenvolvida uma rotina para a criação de cada índice. Os índices serão sempre criados e manipulados em memória principal na inicialização e liberados ao término do programa. Note que o ideal é que os índices primários sejam criados primeiro, depois os secundários.

Após a criação de cada arquivo de índice, deverá ser impresso na tela a frase padrão `INDICE_CRIADO`.

7 Inicialização do programa

Para que o programa inicie corretamente, deve-se realizar o seguinte procedimento:

1. Inserir o comando `SET BTREE_ORDER '<ORDEM DAS ÁRVORES-B>';` para informar a ordem das Árvores-B. Caso não informado, é definido como 3 por padrão;
2. Inserir o comando `SET ARQUIVO_USUARIOS TO '<DADOS DE USUARIOS>';` caso queira inicializar o programa com um arquivo de usuários já preenchido;
3. Inserir o comando `SET ARQUIVO_CURSOS TO '<DADOS DE CURSOS>';` caso queira inicializar o programa com um arquivo de cursos já preenchido;
4. Inserir o comando `SET ARQUIVO_INSCRICOES TO '<DADOS DE INSCRICOES>';` caso queira inicializar o programa com um arquivo de inscrições já preenchido;
5. Inicializar as estruturas de dados dos índices.

8 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 um registro, utilize as funções `exibir_usuario(int rrn)`, `exibir_curso(int rrn)` ou `exibir_inscricao(int rrn)`.

Note que são cinco índices do tipo Árvore-B e uma lista invertida. **Não é necessário implementar rotinas específicas para a manipulação de cada árvore**. Lembre-se de que funções como `bsearch` e `qsort`, por exemplo, são totalmente genéricas e funcionam com qualquer vetor, basta que seja informado o tamanho de cada elemento, o número de posições do vetor e a função de comparação. Use esse mesmo conceito nas funções de manutenção de Árvores-B, lembrando que a única informação que se altera entre elas é o tamanho das chaves.

Implementar rotinas que contenham obrigatoriamente as seguintes funcionalidades:

- Estruturas de dados adequadas para armazenar os índices em arquivos simulados por meio de *strings*;
- Criar os índices primários: deve refazer os índices primários a partir dos arquivos de dados;
- Criar os índices secundários: deve refazer os índices secundários a partir dos arquivos de dados;
- Inserir um registro: modifica os arquivos de dados e de índices;
- Buscar por registro: busca um registro por sua(s) chave(s) primária(a);
- Alterar um registro: modifica o campo do registro diretamente no arquivo de dados;
- Listar registros: listar os registros ordenados pelo intervalo de chaves dadas.

Lembre-se de que, sempre que possível, é **obrigatório o uso da busca binária**, com o arredondamento para **cima** para buscas feitas em índices tanto primários quanto secundários.

9 Resumo de alterações em relação ao T01

Para facilitar o desenvolvimento do T02, é possível reutilizar as funções já implementadas no T01, com ênfase nas seguintes alterações:

1. As operações de remoção de usuário e liberar espaço foram removidas;
2. Todos os índices serão do tipo Árvore-B, menos o índice de lista invertida, que também deve ser alterado para ser armazenado em arquivo;
3. A ordem das Árvores-B é informada na inicialização do programa e é única para todas as árvores;

10 Dicas

- Ao ler uma entrada, tome cuidado com caracteres de quebra de linha (`\n`) não capturados;
- Você nunca deve perder a referência do começo do arquivo, então não é recomendável percorrer a *string* diretamente pelo ponteiro `ARQUIVO`. Um comando equivalente a `fseek(f, 256, SEEK_SET)` é `char *p = ARQUIVO + 256`;
- Diferentemente do `fscanf`, o `sscanf` não movimenta automaticamente o ponteiro após a leitura;
- O `sprintf` e o `strcpy` adicionam 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` e do `strcpy`, não adiciona o caractere nulo no final;
- Atenção ao uso do `sprintf` e do `strcpy`, pois a *string* de destino deve ter tamanho suficiente para também receber o caractere '`\0`'.
- A função `strtok` permite navegar nas *substrings* de uma certa *string* dado 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 as funções auxiliares que estão implementadas no código-base, como por exemplo:
  (a) `int btree_register_size (btree *t)`
  (b) `btree_node btree_node_malloc (btree *t)`
  (c) `void btree_node_free (btree_node no)`
  (d) `char* strpadright (char *str, char pad, unsigned size)`

Com grandes poderes vêm grandes responsabilidades.
— Tio Ben, Spider-man