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

Prazo para entrega: 25/04/2022 – 08:00

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. O que **NÃO** autoriza a cópia de trechos de código, a codificação em conjunto, compra de soluções, ou compartilhamento de tela para resolução do trabalho. Portanto, compartilhem ideias em alto nível, modos de resolver o problema, mas não o código;
- Utilize as mensagens pré-definidas (`#define`).

1 Contexto

Sua plataforma de distribuição de jogos, a **UFSPlay**, se tornou um grande sucesso. Agora, milhares de usuários passaram a usar o seu produto para adquirir tanto jogos nacionais como internacionais. Consequentemente, a quantidade de usuários, jogos e compras 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 jogos são rápidos, mas a manutenção dos índices tem ficado demasiadamente lenta; buscas por compras 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 adequadas, 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

Relembrando, o sistema é composto por dados de usuários, jogos e compras como descrito a seguir

2.1 Dados do usuário

- **ID_User**: identificador único de um usuário (chave primária) que contém 11 números. Não poderá existir outro valor idêntico na base de dados. Ex: `57956238064`;
- **Username**: nome escolhido pelo usuário. Ex: `xXxSN1P3RxXx`;
- **E-mail**: endereço de e-mail do usuário. Ex: `tomas.aquino@gmail.com`;
- **Celular**: número de contato no formato `<99><999999999>`. Ex: `15924835754`;
- **Saldo**: saldo do usuário no formato `<9999999999>.<99>`. Ex: `0000004605.10`;

2.2 Dados do jogo

- **ID_Game**: identificador numérico único para cada jogo, que pode conter até 8 dígitos. Ex: `01234567`;
- **Título**: título do jogo, que deve ser único;
- **Desenvolvedor**: nome do desenvolvedor do jogo;
- **Editora**: nome da editora do jogo;
- **Lançamento**: data em que o jogo foi lançado no formato `<AAAA><MM><DD>`. Ex: `20170224`;
- **Preço**: preço do jogo no formato `<99999999999>.<99>`. Ex: `00000000027.99`

- **Categoria**: até três categorias que o jogo pode pertencer, separadas por '|'. Ex: `Ação|Aventura|Plataforma`

2.3 Dados da compra

- **ID_User**: ID do usuário que realizou a compra do jogo;
- **ID_Game**: ID do jogo comprado;
- **Data**: data em que a compra do jogo foi realizada no formato `<AAAA><MM><DD>`.
  Ex: `20220102`;

Garantidamente, nenhum campo de texto receberá caracteres acentuados.

2.4 Modelo relacional e DDL

A base de dados será mantida em forma de arquivos, porém se fosse observar um modelo relacional, seria equivalente ao observado na Figura 1.

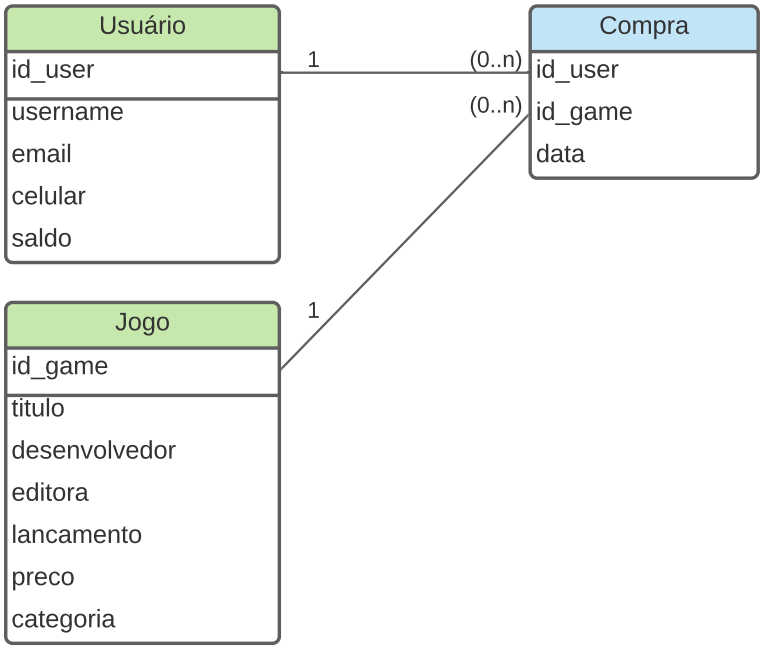


Figura 1: Modelo relacional das tabelas de usuários, jogos e compras da UFSPlay

```
CREATE TABLE usuarios (
  id_user   varchar(11) NOT NULL PRIMARY KEY,
  username  text NOT NULL,
  email     text NOT NULL,
  celular   varchar(11),
  saldo     numeric(12, 2) DEFAULT 0,
);
```

```
CREATE TABLE jogos (
  id_game        varchar(8) NOT NULL PRIMARY KEY,
  titulo         text NOT NULL UNIQUE,
  desenvolvedor  text NOT NULL,
  editora        text NOT NULL,
  lancamento     varchar(8) NOT NULL,
  preco          numeric(12, 2) NOT NULL,
  categoria      text[3] DEFAULT '{}'
);
CREATE TABLE compras (
  id_user  varchar(11) NOT NULL,
  id_game  varchar(8) NOT NULL,
  data     varchar(8) NOT NULL,
);
```

3 Operações suportadas pelo programa

Deve ser possível interagir com o programa 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 usuário

```
INSERT INTO usuarios VALUES ('<id_user>', '<username>', '<email>');
```

Para criar uma nova conta de usuário, seu programa deve ler os campos `id_user`, `username` e `email`. Inicialmente, a conta será criada sem saldo (R$0,00) e sem um número de celular, sendo que este deverá ser preenchido com '*' (asterisco) para todos os dígitos do número. 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_User já cadastrado, ou seja, com ID_User 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 Cadastro de jogo

```
INSERT INTO jogos VALUES ('<titulo>', '<desenvolvedor>', '<editora>', '<lanca-
mento>', '<preco>');
```

O jogo é adicionado no banco de dados. Seu programa deve ler os campos que contém o `título`, `desenvolvedor`, `editora`, `lançamento` e `preço`. Assim como no cadastro de usuário, todos os campos serão fornecidos de maneira regular, não havendo a necessidade de pré-processamento. O ID_Game do jogo deverá ser preenchido de acordo com a quantidade de jogos cadastrados no sistema, ou seja, é

um valor incremental. A função deve falhar caso haja a tentativa de inserir um jogo cujo título já esta 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`.

3.3 Compra de jogo

```
INSERT INTO compras VALUES ('<id_user>', '<titulo>');
```

O usuário poderá comprar um jogo dado o ID_User do usuário e o título do jogo. Caso o jogo ou o usuário não existam, o programa deve imprimir `ERRO_REGISTRO_NAO_ENCONTRADO`. Caso a compra já tenha sido realizada, o seu programa deve imprimir `ERRO_PK_REPETIDA`. Primeiro, é preciso checar se o jogo e o usuário existem. Caso o saldo presente na conta do usuário seja insuficiente para comprar o jogo, ela não deve ser efetuada e a mensagem padrão `ERRO_SALDO_NAO_SUFICIENTE` deve ser imprimida. Caso não haja nenhum desses problemas, a compra deve ser realizada, gravando a data em que a foi feita, atualizando os índices necessários e imprimindo a mensagem padrão `SUCESSO`.

Lembre-se de atualizar todos os índices necessários após a inserção da compra.

3.4 Cadastro de celular

```
UPDATE usuarios SET celular = '<celular>' WHERE id_user = '<id_user>';
```

O usuário deverá poder atualizar seu número de celular dado o ID_User 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`.

3.5 Adicionar saldo a conta

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

O usuário deverá ser capaz de adicionar dinheiro na sua conta dado seu ID_User 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`.

3.6 Adicionar categoria ao jogo

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

O usuário deve poder adicionar uma categoria a um jogo dado seu título e a categoria escolhida. Caso o jogo não exista, o programa deve imprimir `ERRO_REGISTRO_NAO_ENCONTRADO`

e caso a categoria nova já esteja presente nas categorias do jogo, seu programa deve imprimir `ERRO_CATEGORIA_REPETIDA`. Existe um máximo de três categorias por jogo e, garantidamente, não haverá nenhuma tentativa de inserir mais de três por jogo. Caso não haja nenhum erro, o programa deve atribuir a categoria ao jogo, atualizando todos os índices e arquivos necessários e então imprimir a mensagem padrão `SUCESSO`.

3.7 Busca

O usuário deverá buscar usuários e jogos pela(s) chave(s).

Note que a utilização da **busca binária** com arredondamento **para cima** é sempre necessária.

Além disso, 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. As seguintes operações de busca por usuários e jogos deverão ser implementadas:

3.7.1 Usuários

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

(a) Por ID_User de usuário:

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

Solicitar ao usuário o ID_User vinculado a conta. 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.

3.7.2 Jogos

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

(a) Pelo ID_Game do jogo:

```
SELECT * FROM jogos WHERE id_game = '<id_game>';
```

Solicitar ao usuário o ID_Game do jogo no formato correto de 8 dígitos. Caso o jogo não exista, seu programa deverá exibir a mensagem padrão `ERRO_REGISTRO_NAO_ENCONTRADO`. Caso o jogo exista, todos os dados do jogo deverão ser impressos na tela de forma formatada.

(b) Pelo título do jogo:

```
SELECT * FROM jogos WHERE titulo = '<titulo do jogo>';
```

Solicitar ao usuário o título do jogo que deseja ser encontrado. Caso o título do jogo não seja encontrado, seu programa deverá exibir a mensagem padrão `ERRO_REGISTRO_NAO_ENCONTRADO`. Caso o jogo exista, todos os dados deverão ser impressos na tela de forma formatada.

3.8 Listagem

As seguintes operações de listagem de usuários, jogos e compras deverão ser implementadas.

3.8.1 Usuários

(a) Por ID_User dos usuários:

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

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

3.8.2 Jogos

(a) Por categoria:

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

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

Antes de listar os jogos, 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 secundários percorridos:   2 1 0
 Registros primários percorridos:   3 5 7
```

3.8.3 Compras

(a) Por data de aquisição do jogo:

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

Exibe todas as compras realizada em um determinado período de tempo (`data` entre `<data início>` e `<data fim>`), em ordem cronológica. 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`.

3.9 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 jogos:
`\echo file ARQUIVO_JOGOS`

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

(c) Dados das compras:
`\echo file ARQUIVO_COMPRAS`

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

3.10 Imprimir arquivos de índice

O sistema deverá imprimir os arquivos de índices 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 jogos:
`\echo index jogos_idx`

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

(c) Índice de compras:
`\echo index compras_idx`

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

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

Imprime o arquivo de índice secundário com o título dos jogos e os respectivos ID_Game (`jogos_idx`). Caso o arquivo esteja vazio, imprimir `ERRO_ARQUIVO_VAZIO`.

(e) Índice de datas de compras com `data`, `id_user` e `id_game`:
`\echo index data_user_game_idx`

Imprime o arquivo de índice secundário para a data, o ID_User do dono da compra e o ID_Game do jogo (`data_user_game_idx`). Caso o arquivo esteja vazio, imprimir `ERRO_ARQUIVO_VAZIO`.

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

Imprime o arquivo de índice primário com o ID_Game dos jogos das categorias (`categorias_primario_idx`) e o RRN do próximo jogo dessa categoria. Caso o arquivo esteja vazio, imprimir `ERRO_ARQUIVO_VAZIO`.

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

Imprime o arquivo de índice secundário das categorias dos jogos (`categorias_secundario_idx`) que contém o nome da categoria e o RRN do primeiro jogo da categoria no índice primário. Caso o arquivo esteja vazio, imprimir `ERRO_ARQUIVO_VAZIO`.

3.11 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_JOGOS` e `ARQUIVO_COMPRAS` 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 `username` (tamanho máximo de 47 *bytes*) e `email` (tamanho máximo de 41 *bytes*) devem ser de tamanho variável. Os demais campos devem ser de tamanho fixo, sendo eles `id_user` (11 *bytes*), `celular` (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:

```
66545678765;SnailThemi;mariaeugenia@gmail.com;**********;00000
00030.00;##################################################
##99876556782;ezPz#Fireclan;ga.augusto@gmail.com;**********;00
00000000.00;##################################################
####44565434213;forninhocaiu;ge.santana@gmail.com;**********;0
000000029.69;##################################################
######54654367865;batata_doce;galmeida@gmail.com;**********;00
00001250.80;##################################################
########37567876542;Erlkonigin;melissa@gmail.com;**********;10
00000440.92;##################################################
##########
```

`ARQUIVO_JOGOS`: o arquivo de jogos deverá ser organizado em registros de tamanho fixo de 256 *bytes* (256 caracteres). Os campos `titulo` (máximo de 43 *bytes*), `desenvolvedor` (máximo de 47 *bytes*), `editora` (máximo de 47 *bytes*) e `categoria` (máximo de 60 *bytes*, com no máximo 3 valores, onde cada categoria pode ter no máximo 20 *bytes*) devem ser de tamanhos variáveis. O campo multivalorado `categoria` deve ter os seus valores separados por '`|`'. Os demais campos são de tamanho fixo e possuem as seguintes especificações: `id_game` (8 *bytes*), `lancamento` (8 *bytes*) e `preco` (13 *bytes*), totalizando 29 *bytes* de tamanho fixo. Assim como no registro de usuários, os campos devem ser separados pelo delimitador '`;`', cada registro terá 7 delimitadores para os campos e mais 3 para o campo multivalorado se necessário e, caso o registro tenha menos que 256 *bytes*, o espaço restante deverá ser preenchido com o caractere '`#`'. Como são 29 *bytes* de tamanho fixo + 7 *bytes* para os delimitadores, os campos variáveis devem ocupar no máximo 220 *bytes* para que não se exceda o tamanho do registro.

Exemplo de arquivo de dados de jogos:

```
00000000;Meia-Vida;Valvula;Valvula;19981119;0000000029.99;FPS;###
############################################################
############################################################
############################################################0000
0001;Presa;Cabeca de Melao Studios;40K Martelos;20070711;00000000
44.29;;##################################################################
##################################################################
##################################################################00000002
;Quinze Minutos;Antonio Luis;Annapurr;20210819;0000000051.99;;###
##################################################################
##################################################################
##################################################################00000003;Esq
uerda 4 Morto 1;Valvula;Valvula;20091117;0000000020.69;FPS;######
##################################################################
##################################################################
##################################################00000004;Cavalei
ro Vazio;Time Cereja;Time Cereja;20170224;0000000027.99;Metroidva
nia|Plataforma|Acao;############################################
##################################################################
##############################################
```

ARQUIVO_COMPRAS: o arquivo de compras deverá ser organizado em registros de tamanho fixo de 27 *bytes*. Todos os campos possuem um tamanho fixo e, portanto, não é necessário a presença de delimitadores: `id_user` (11 *bytes*), `id_game` (8 *bytes*) e `data` (8 *bytes*). No exemplo abaixo, os campos estão ordenados por `id_user`, `data` e `id_game`.

Exemplo de arquivo de dados de compras:

```
44565434213 20210920 00000000
37567876542 20210924 00000004
37567876542 20211003 00000008
66545678765 20211005 00000005
```

Como os campos de compras possuem tamanho fixo, não são necessários delimitadores (foram inseridos espaços no exemplo acima para maior clareza). Note também, que não há quebras de linhas nos arquivos (elas também foram inseridas no exemplo apenas para facilitar a visualização da sequência de registros).

5 Instruções para as operações com os registros

- **Inserção**: cada usuário, jogo e compra devem ser inseridos no final de seus respectivos arquivos de dados, e atualizados os índices.

- **Atualização**: existem 2 campos alteráveis, o saldo e o celular do usuário. Todos eles possuem seus tamanhos pré-determinados, 13 *bytes* para o saldo e 11 *bytes* para o celular. Uma vez que haja a atualização de algum desses campos, eles devem ser atualizados no registro de mesma posição que está (não deve se remover e em seguida inserir o registro).

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) e **a promoção deverá ser sempre pelo sucessor imediato** (menor chave da sub-árvore à direita). Todo novo nó criado deverá ser inserido no final do respectivo arquivo de índice.

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_User 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: `43487689087`, `52587909876` e `21046578965` (nesta ordem).

- Inserindo a chave `43487689087`:

Em disco, o arquivo de índice primário seria:

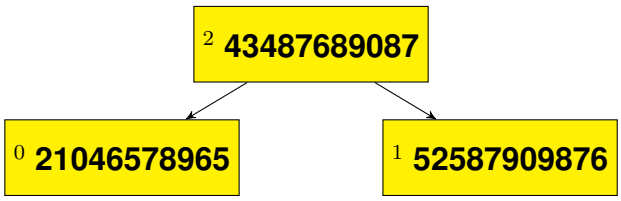
```
001 | 43487689087 0000 | ########## #### | T | *** *** ***
```

- Inserindo a chave 52587909876:

Em disco:

```
002 | 43487689087 0000 | 52587909876 0001 | T | *** *** ***
```

- Inserindo a chave 21046578965:



Ao inserir a chave iniciada por 21046578965, 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 | 21046578965 0002 | ########## #### | T | *** *** ***
001 | 52587909876 0001 | ########## #### | T | *** *** ***
001 | 43487689087 0000 | ########## #### | F | 000 001 ***
```

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

```
{QUANTIDADE DE CHAVES}
{ID_USER_1} {RRN_1}
{ID_USER_2} {RRN_2}
...
{ID_USER_(m-1)} {RRN_(m-1)}
{FOLHA}
{RRN FILHO_1} {RRN FILHO_2} ... {RRN_FILHO_m}
```

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

6.1.2 `jogos_idx`

Índice primário do tipo Árvore-B que contém o ID_Game 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 `usuarios_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.

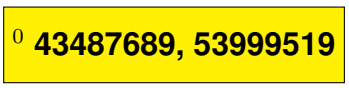
Exemplo da representação da Árvore-B de ordem 3, após a inserção das chaves: `43487689`, `53999519` e `39440633` (nesta ordem).

- Inserindo a chave `43487689`:

Em disco, o arquivo de índice primário seria:

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

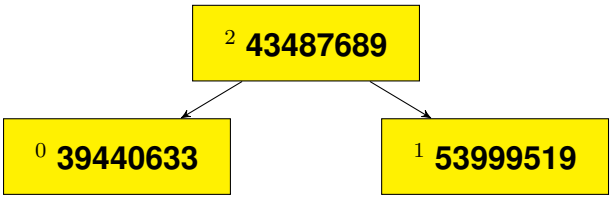
- Inserindo a chave `53999519`:



Em disco:

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

- Inserindo a chave `39440633`:



Ao inserir a chave iniciada por `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 ID_Game, de ordem m, possui as seguintes informações:

```
{QUANTIDADE DE CHAVES}
{ID_GAME_1} {RRN_1}
{ID_GAME_2} {RRN_2}
...
{ID_GAME_(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 `compras_idx`

Índice primário do tipo Árvore-B que contém o ID_User do usuário que realizou a compra (em ordem crescente), o ID_Game do jogo (também em ordem crescente) e o RRN do respectivo registro no arquivo de compras.

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

- *Contador de chaves:* 3 bytes para a quantidade de chaves;
- *Chaves ordenadas:* $(m-1)$ * (19 bytes de chave primária [11 bytes para o ID_User + 8 bytes para o ID_Game] + 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 compras representado por uma Árvore-B de ordem 4, após a inserção das chaves na ordem: ‘`44678965437 25091500 0002`’, ‘`34678965321 20210912 0000`’ e ‘`51478965098 20210101 0001`’.

[0] **346..., 446..., 514...**

```
003 | 34678965321 20210912 0002 | 44678965437 25091500 0000 |
      51478965098 20210101 0001 | T | *** *** *** ***
```

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

```
{QUANTIDADE DE CHAVES}
{ID_USER_1} {ID_GAME_1} {RRN_1}
{ID_USER_2} {ID_GAME_2} {RRN_2}
...
{ID_USER_(m-1)} {ID_GAME_(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 o título de todos os jogos (em ordem lexicográfica) e a chave primária do respectivo jogo.

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:* $(m-1)$ * (44 bytes de título + 8 bytes de ID_Game do jogo) 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 '`Cavaleiro do Vazio 46578965`'.


```
001 | Cavaleiro do Vazio########################## 46578965 |
      ############################################ ######## |
      ############################################ ######## |
      ############################################ ######## |
      T | *** *** *** *** ***
```

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

```
{QUANTIDADE DE CHAVES}
{TITULO_1} {ID_GAME_1}
{TITULO_2} {ID_GAME_2}
...
{TITULO_(m-1)} {ID_GAME_(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_user_game_idx`

Índice do tipo Árvore-B que associa as datas das compras com os usuários e jogos (em ordem cronológica).

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

- *Contador de chaves:* 3 bytes para a quantidade de chaves;
- *Chaves ordenadas:* $(m-1)$ * (8 bytes para data + 19 bytes de chave primária [11 bytes para o ID_User + 8 bytes para o ID_Game]) 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 primário representado por uma Árvore-B de ordem 4, após a inserção das chaves na ordem: ‘`20210325 44678965437 00000001`’, ‘`20210912 34678965321 01234567`’ e ‘`20210101 51478965098 00000002`’.


```
003 | 20210101 51478965098 00000002 | 20210912 34678965321 01234567 |
      20210325 44678965437 00000001 | T | *** *** *** ***
```

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

```
{QUANTIDADE DE CHAVES}
{DATA_1} {ID_USER_1} {ID_GAME_1}
{DATA_2} {ID_USER_2} {ID_GAME_2}
...
{DATA_(m-1)} {ID_USER_(m-1)} {ID_GAME_(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 o ID_Game dos jogos presentes e o apontador para o próximo jogo de mesma categoria. Se não houver um próximo jogo, esse apontador deve ser igual à -1. No índice secundário estão as categorias, assim como a referência do primeiro jogo daquela categoria no índice primário. Para padronizar o índice secundário, complete a categoria com ‘`#`’ para ocupar exatamente 18 bytes. Da mesma forma, preencha os apontadores para ocuparem 4 bytes e no caso do -1 represente como ‘`-001`’. Para ilustrar esses índices, considere o seguinte exemplo:

Categorias primário	
ID_Game	próx. registro
0	4
4	-1
4	7
4	5
3	-1
7	6
5	-1
5	-1

Categorias secundário	
Categoria	registro
Acao	3
FPS	0
Metroidvania	1
Plataforma	2

No exemplo acima, podemos observar que a tabela de categorias secundário possui a categoria na primeira coluna, assim como o RRN do primeiro jogo daquela categoria que foi inserido na tabela de categorias primário. Na tabela primária, temos na primeira coluna o ID_Game dos jogos em cada categoria. Note que o jogo 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 jogo de mesma categoria na própria tabela de categorias primária, sendo que, no caso de RNN $= -1$, significa que aquele jogo é o último de sua categoria. Vale destacar que o índice primário **não** precisa estar ordenado, 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:
Acao############## 0003 FPS############### 0000
Metroidvania###### 0001 Plataforma######## 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 utilizando os pseudo-arquivos de índices na inicialização e liberados ao término do programa. Note que o ideal é que os índices primários sejam criados primeiro, depois os secundários.

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 '<DADOS DE USUARIOS>';` para informar os dados contidos no arquivo de usários ou `SET ARQUIVO_USUARIOS '';` caso o arquivo esteja vazio;
3. Inserir o comando `SET ARQUIVO_JOGOS '<DADOS DE JOGOS>';` para informar os dados contidos no arquivo de jogos ou `SET ARQUIVO_JOGOS '';` caso o arquivo esteja vazio;
4. Inserir o comando `SET ARQUIVO_COMPRAS '<DADOS DE COMPRAS>';` para informar os dados contidos no arquivo de compras ou `SET ARQUIVO_COMPRAS '';` caso o arquivo esteja vazio;
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_jogo(int rrn)` ou `exibir_compra(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, tanto na busca dentro dos nós da arvore-B quanto na busca pelos jogos nas categorias da lista invertida, é **obrigatório o uso da busca binária, com o arredondamento para cima**.

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.

Mas no final é algo que passará, essa sombra, até mesmo a escuridão acabará. Um novo dia virá. E quando o sol nascer ele brilhará ainda mais. Essas eram as histórias que ficavam com a gente, que significavam alguma coisa, mesmo quando eu era pequeno demais para entender porque. Mas eu acho, sr. Frodo, que eu entendo. As pessoas daquelas histórias tiveram muitas chances para desistir, mas não desistiram. Elas foram em frente porque estavam se agarrando a alguma coisa.

— *Samwise Gamgee, O Retorno do Rei*