Universidade de São Paulo
Instituto de Ciências Matemáticas e de Computação
Departamento de Ciências de Computação
Disciplina de Estrutura de Dados III (SCC0607)

Docente
Profa. Dra. Cristina Dutra de Aguiar
cdac@icmc.usp.br

Monitores
Gabriel Vicente Rodrigues
gabriel_vr@usp.br *ou telegram:@ga_vr*
Lucas de Medeiros Franca Romero
lucasromero@usp.br *ou telegram: @lucasromero*

Voluntário
João Paulo Clarindo
jpcsantos@usp.br

Recuperação

Esta recuperação tem como objetivo indexar arquivos de dados usando um índice árvore-B.

A recuperação deve ser feita individualmente. A solução deve ser proposta exclusivamente pelo aluno com base nos conhecimentos adquiridos nas aulas. Consulte as notas de aula e o livro texto quando necessário.

Descrição do arquivo de índice árvore-B

O índice árvore-B com ordem m é definido formalmente como descrito a seguir.

1. Cada página (ou nó) do índice árvore-B deve ser, pelos menos, da seguinte forma:

   $< <C_1,P_{R1}>, <C_2,P_{R2}>, ..., <C_{q-1},P_{Rq-1}>, P_1, P_2, ..., P_{q-1}, P_q>$, onde $(q \leq m)$ e

   - Cada C_i $(1 \leq i \leq q-1)$ é uma chave de busca.
   - Cada P_{Ri} $(1 \leq i \leq q-1)$ é um campo de referência para o registro no arquivo de dados que contém o registro de dados correspondente a C_i.
   - Cada P_j $(1 \leq j \leq q)$ é um campo de referência para uma subárvore ou assume o valor -1 caso não exista subárvore (ou seja, caso seja um nó folha).

2. Dentro de cada página (ou seja, as chaves de busca são ordenadas)
   - $C_1 < C_2 < ... < C_{q-1}$.
3. Para todos os valores X da chave na subárvore apontada por P_i:
   - $C_{i-1} < X < C_i$ para $1 < i < q$
   - $X < C_i$ para $i = 1$
   - $C_{i-1} < X$ para i = q.
4. Cada página possui um máximo de m descendentes.
5. Cada página, exceto a raiz e as folhas, possui no mínimo $\lceil m/2 \rceil$ descendentes (*taxa de ocupação*).
6. A raiz possui pelo menos 2 descendentes, a menos que seja um nó folha.
7. Todas as folhas aparecem no mesmo nível.
8. Uma página não folha com k descendentes possui k-1 chaves.
9. Uma página folha possui no mínimo $\lceil m/2 \rceil - 1$ chaves e no máximo m – 1 chaves (*taxa de ocupação*).

Descrição do Registro de Cabeçalho. O registro de cabeçalho deve conter os seguintes campos:

- *status*: indica a consistência do arquivo de índice, devido à queda de energia, travamento do programa, etc. Pode assumir os valores 0, para indicar que o arquivo de dados está inconsistente, ou 1, para indicar que o arquivo de dados está consistente. Ao se abrir um arquivo para escrita, seu *status* deve ser 0 e, ao finalizar o uso desse arquivo, seu *status* deve ser 1 – tamanho: *string* de 1 byte.
- *noRaiz*: armazena o RRN do nó (página) raiz do índice árvore-B. Quando a árvore-B está vazia, noRaiz = -1 – tamanho: inteiro de 4 *bytes*
- *nroNiveis:* armazena o número de níveis do índice árvore-B. Inicialmente, a árvore-B está vazia e, portanto, *nroNiveis* = 0. Quando as primeiras *m* chaves de busca são inseridas, o nó raiz é igual ao nó folha e, nesse caso, *nroNiveis* = 1. Quando ocorrer o primeiro *split*, *nroNiveis* = 2. Depois disso, cada vez que o

nível da árvore aumentar, *nroNiveis* deve ser incrementado – tamanho: inteiro de 4 *bytes*

- *proxRRN:* armazena o valor do próximo RRN a ser usado para conter um nó (página da árvore-B). Inicialmente, a árvore-B está vazia e, portanto, *proxRRN* = 0. Quando o primeiro nó é criado (nó folha = nó raiz), *proxRRN* = 1. Depois, quando primeiro *split* acontece, *proxRRN* = 3. A cada nó criado da árvore-B, *proxRRN* é incrementado – tamanho: inteiro de 4 *bytes*
- *nroChaves:* armazena o número de chaves de busca indexadas no índice árvore-B. Inicialmente, a árvore-B está vazia e, portanto, *nroChaves* = 0. A cada chave de busca inserida na árvore-B, *nroChaves* é incrementado – tamanho: inteiro de 4 *bytes*

Representação Gráfica do Registro de Cabeçalho. O tamanho do registro de cabeçalho é de 72 bytes, representado da seguinte forma:

1 byte	4 bytes				4 bytes				4 bytes				4 bytes				55 bytes	
status	*noRaiz*				*nroNiveis*				*proxRRN*				*nroChaves*				*lixo (caractere '$')*	
0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*	*16*	*...*	*71*

Observações Importantes.

- O registro de cabeçalho deve seguir estritamente a ordem definida na sua representação gráfica.
- Os nomes dos atributos também devem seguir estritamente os nomes definidos na especificação dos mesmos.
- Para seguir a especificação do conceito de árvore-B, o nó da árvore-B deve obrigatoriamente ser do tamanho de uma página de disco. Entretanto, isso não será seguido neste trabalho para simplificar a quantidade de chaves de busca que são armazenadas no nó. Lembrando também que as páginas de disco têm potência de 2, o que também não será seguido neste trabalho por simplificação.
- Os 59 bytes restantes devem ser preenchidos com lixo. O lixo é representado pelo caractere '$'.

Descrição do Registro de Dados. Deve ser considerada a seguinte organização: campos de tamanho fixo e registros de tamanho fixo. Em adição ao Item 1 da definição formal do índice árvore-B, cada nó (página) da árvore também deve armazenar dois outros campos:

- *nível*, indicando o nível no qual o nó se encontra. Quando um nó é um nó-folha, *nível* = 1. Quando um nó aponta para um nó-folha, *nível* = 2. Quando um nó aponta para um outro nó que, por sua vez, aponta para um nó-folha, *nível* = 3. E assim sucessivamente – tamanho: inteiro de 4 *bytes*
- *n*, indicando o número de chaves presentes no nó – tamanho: inteiro de 4 *bytes*.

A ordem da árvore-B é 6, ou seja, $m = 6$. Portanto, um nó (página) terá 5 chaves e 6 descendentes. A chave de busca é o campo *codEstacao*.

Considere que deve ser implementada a rotina de *split* durante a inserção. Considere que a distribuição das chaves de busca deve ser o mais uniforme possível. Considere também que a chave de busca a ser promovida deve ser a primeira chave do novo nó resultante do particionamento (ou seja, o primeiro elemento do segundo nó é a chave promovida durante o particionamento). Quando necessário, o nó mais à esquerda deverá conter uma chave de busca a mais. Considere que a página sendo criada é sempre a página à direita.

Representação Gráfica de um Nó (Página/Registro de Dados) do índice. O tamanho do registro de cabeçalho é de 72 bytes, representado da seguinte forma:

4 bytes				4 bytes				4 bytes	4 bytes		4 bytes	4 bytes	4 bytes	4 bytes	4 bytes				
nível				*n*				C_1	P_{R1}	...	C_5	P_{R5}	P_1	...	P_6				
0	1	2	3	4	5	6	7	...							68	69	70	71	

Observações Importantes.

- Cada registro de dados deve seguir estritamente a ordem definida na sua representação gráfica.
- Os nomes dos atributos também devem seguir estritamente os nomes definidos na especificação dos mesmos.

- Quando um nó (página) do índice tiver chaves de busca que não forem preenchidas, a chave de busca deve ser representada pelo valor -1 e o ponteiro para o arquivo de dados deve ser representado pelo valor -1. O valor -1 deve ser usado para denotar que um ponteiro P_i ($1 \leq i \leq m$) de um nó da árvore-B que é nulo.

Programa

Descrição Geral. Desenvolva um programa usando a linguagem C de forma que ele possa realizar operações sobre um índice árvore-B e que possa *inserir* e *buscar* dados de um arquivo de dados que está indexado por esse índice.

Importante. A definição da sintaxe de cada comando bem como sua saída devem seguir estritamente as especificações definidas em cada funcionalidade. Para especificar a sintaxe de execução, considere que o programa seja chamado de "programaTrab". Essas orientações devem ser seguidas uma vez que a correção do funcionamento do programa se dará de forma automática. De forma geral, a primeira entrada da entrada padrão é sempre o identificador de suas funcionalidades, conforme especificado a seguir.

Descrição Específica. O programa deve oferecer as seguintes funcionalidades:

[12] Crie um arquivo de índice árvore-B para um arquivo de dados de entrada já existente, que é o arquivo de dados definido de acordo com a especificação do primeiro trabalho prático, e que pode conter registros logicamente removidos. O campo a ser indexado é *codEstacao*. Registros logicamente removidos presentes no arquivo de dados de entrada não devem ter suas chaves de busca correspondentes no arquivo de índice. A inserção no arquivo de índice deve ser feita um-a-um. Ou seja, para cada registro não removido presente no arquivo de dados, deve ser feita a inserção de sua chave de busca correspondente no arquivo de índice árvore-B. A manipulação do arquivo de índice árvore-B deve ser feita em disco, de acordo com o conteúdo ministrado em sala de aula. Antes de terminar a execução da funcionalidade, deve ser utilizada a função binarioNaTela, disponibilizada na página do projeto da disciplina, para mostrar a saída do arquivo de índice árvore-B.

Entrada do programa para a funcionalidade [12]:

```
12 arquivoEntrada.bin arquivoIndice.bin
```

onde:

- `arquivoEntrada.bin` é um arquivo binário de entrada que segue as mesmas especificações do primeiro trabalho prático, e que contém dados desordenados e registros logicamente removidos.
- `arquivoIndice.bin` é o arquivo binário de índice árvore-B que indexa o campo *codEstacao*. Esse arquivo deve seguir as especificações definidas neste trabalho prático.

Saída caso o programa seja executado com sucesso:

```
Listar o arquivo binário arquivoIndice.bin usando a função binarioNaTela.
```

Mensagem de saída caso algum erro seja encontrado:

```
Falha no processamento do arquivo.
```

Exemplo de execução:

```
./programaTrab
12 arquivoEntrada.bin arquivoIndice.bin
usar a função binarioNaTela antes de terminar a execução da funcionalidade, para mostrar a saída do arquivo arquivoIndice.bin, o qual indexa o campo codEstacao.
```

[13] Permita a recuperação dos dados de todos os registros que satisfaçam um critério de busca determinado pelo usuário sobre o campo *codEstacao*, usando o índice árvore-B criado na funcionalidade [12]. Note que somente o campo *codEstacao* deve ser utilizado como forma de busca, desde que o índice criado na funcionalidade [12] indexa chaves de busca desse campo. Essa funcionalidade pode retornar 0 registros (quando nenhum satisfaz ao critério de busca) ou 1 registro (desde que o identificador do servidor não aceita valores repetidos). A manipulação do arquivo de índice árvore-B deve ser feita em disco, de acordo com o conteúdo ministrado em sala de aula. Os dados devem ser mostrados no mesmo formato definido para a funcionalidade [3]. Registros marcados como logicamente removidos não devem ser exibidos.

Sintaxe do comando para a funcionalidade [13]:

```
13 arquivoEntrada.bin arquivoIndice.bin codEstacao valor
```

onde:

- `arquivoEntrada.bin` é um arquivo binário de entrada que segue as mesmas especificações do primeiro trabalho prático, e que contém dados desordenados e registros logicamente removidos.
- `arquivoIndice.bin` é o arquivo binário de índice árvore-B que indexa o campo *codEstacao*. Esse arquivo deve seguir as especificações definidas neste trabalho prático.
- `codEstacao` é o nome do campo que representa a chave de busca.
- `valor` é o número que representa o `codEstacao` sendo procurado.

Saída caso o programa seja executado com sucesso:

Cada registro deve ser mostrado em uma única linha e os seus campos devem ser mostrados de forma sequencial separado por um espaço em branco. Campos de tamanho fixo que tiverem o valor nulo devem ser exibidos da seguinte forma: ao invés de exibir o valor -1, escreva NULO. Campos de tamanho variável que tiverem o valor nulo devem ser exibidos da seguinte forma: NULO. A ordem de exibição dos campos dos registros deve ser codEstacao, nomeEstacao, codLinha, nomeLinha, codProxEstacao, distProxEstacao, codLinhaIntegra, codEstIntegra, conforme ilustrado no **exemplo de execução.**

Mensagem de saída caso não seja encontrado o registro que contém o valor do campo ou o campo pertence a um registro que esteja removido:

```
Registro inexistente.
```

Mensagem de saída caso algum erro seja encontrado:

```
Falha no processamento do arquivo.
```

Exemplo de execução (é mostrado um registro ilustrativo):

```
./programaTrab
13 arquivoEntrada.bin arquivoIndice.bin codEstacao 55
55 Luz 4 Amarela 56 1257 1 9
```

[14] Estenda a funcionalidade [5] de forma que, a cada inserção de um registro no arquivo de dados, a chave de busca correspondente a essa inserção seja inserida no arquivo de índice árvore-B. Antes de terminar a execução da funcionalidade, deve ser utilizada a função binarioNaTela, disponibilizada na página do projeto da disciplina, para mostrar a saída do arquivo de índice árvore-B.

Entrada do programa para a funcionalidade [14]:

`14 arquivoEntrada.bin arquivoIndice.bin n`

`codEstacao`$_1$ `nomeEstacao`$_1$ `codLinha`$_1$ `nomeLinha`$_1$ `codProxEstacao`$_1$ `distProxEstacao`$_1$ `codLinhaIntegra`$_1$ `codEstacaoIntegra`$_1$

`codEstacao`$_2$ `nomeEstacao`$_2$ `codLinha`$_2$ `nomeLinha`$_2$ `codProxEstacao`$_2$ `distProxEstacao`$_2$ `codLinhaIntegra`$_2$ `codEstacaoIntegra`$_2$

`...`

`codEstacao`$_n$ `nomeEstacao`$_n$ `codLinha`$_n$ `nomeLinha`$_n$ `codProxEstacao`$_n$ `distProxEstacao`$_n$ `codLinhaIntegra`$_n$ `codEstacaoIntegra`$_n$

onde:

- `arquivoEntrada.bin` é um arquivo binário de entrada que segue as especificações definidas no primeiro trabalho prático. As inserções a serem realizadas nessa funcionalidade devem ser feitas nesse arquivo.
- `arquivoIndice.bin` é o arquivo binário de índice árvore-B que indexa o campo *codEstacao*. Esse arquivo deve seguir as especificações definidas neste trabalho prático.
- `n` é o número de inserções a serem realizadas. Para cada inserção, deve ser informado os valores a serem inseridos no arquivo, considerando os seguintes campos, na seguinte ordem: codEstacao, nomeEstacao, codLinha, nomeLinha, codProxEstacao, distProxEstacao, codLinhaIntegra, codEstIntegra. Valores nulos devem ser identificados, na entrada da funcionalidade, por NULO. Cada uma das `n` inserções deve ser especificada em uma linha diferente. Deve ser deixado um espaço em branco entre os valores dos campos. Os valores dos campos do tipo *string* devem ser especificados entre aspas duplas (").

Saída caso o programa seja executado com sucesso:

`Listar o arquivo binário arquivoIndice.bin usando a função binarioNaTela.`

Mensagem de saída caso algum erro seja encontrado:

`Falha no processamento do arquivo.`

Exemplo de execução:

```
./programaTrab
14 arquivoEntrada.bin arquivoIndice.bin 2
500 "Teste" 10 "Branca" NULO NULO NULO
501 "Nova Estacao" 10 "Branca" NULO NULO NULO NULO
```

`usar a função binarioNaTela antes de terminar a execução da funcionalidade, para mostrar a saída do arquivo arquivoEntrada.bin, o qual foi atualizado com as inserções.`

Restrições

As seguintes restrições têm que ser garantidas no desenvolvimento do trabalho.

[1] O arquivo de dados deve ser gravado em disco no **modo binário**. O modo texto não pode ser usado.

[2] Os dados do registro descrevem os nomes dos campos, os quais não podem ser alterados. Ademais, todos os campos devem estar presentes na implementação, e nenhum campo adicional pode ser incluído. O tamanho e a ordem de cada campo deve obrigatoriamente seguir a especificação.

[3] Deve haver a manipulação de valores nulos, conforme as instruções definidas.

[4] Não é necessário realizar o tratamento de truncamento de dados.

[5] Devem ser exibidos avisos ou mensagens de erro de acordo com a especificação de cada funcionalidade.

[6] Os dados devem ser obrigatoriamente escritos campo a campo. Ou seja, não é possível escrever os dados registro a registro. Essa restrição refere-se à entrada/saída, ou seja, à forma como os dados são escritos no arquivo. Para se fazer a busca, é possível caminhar no arquivo registro a registro, já que se sabe o tamanho do registro.

[7] O(s) aluno(s) que desenvolveu(desenvolveram) o trabalho prático deve(m) constar como comentário no início do código (i.e. NUSP e nome do aluno). Para trabalhos desenvolvidos por mais do que um aluno, não será atribuída nota ao aluno cujos dados não constarem no código fonte.

[8] Todo código fonte deve ser documentado. A **documentação interna** inclui, dentre outros, a documentação de procedimentos, de funções, de variáveis, de partes do

código fonte que realizam tarefas específicas. Ou seja, o código fonte deve ser documentado tanto em nível de rotinas quanto em nível de variáveis e blocos funcionais.

[9] A implementação deve ser realizada usando a linguagem de programação C. As funções das bibliotecas <stdio.h> devem ser utilizadas para operações relacionadas à escrita e leitura dos arquivos. A implementação não pode ser feita em qualquer outra linguagem de programação. O programa executará no [run.codes].

Fundamentação Teórica

Conceitos e características dos diversos métodos para representar os conceitos de campo e de registro em um arquivo de dados podem ser encontrados nos *slides* de sala de aula e também nas páginas 96 a 107 do livro *File Structures* (*second edition*), de Michael J. Folk e Bill Zoellick.

Material para Entregar

Arquivo compactado. Deve ser preparado um arquivo .zip contendo:

- Código fonte do programa devidamente documentado.
- Makefile para a compilação do programa.
- Um vídeo gravado pelo aluno que fez a recuperação, o qual deve ter, no máximo, 5 minutos de gravação. O vídeo deve explicar o trabalho desenvolvido. Ou seja, o aluno deve apresentar: cada funcionalidade e uma breve descrição de como a funcionalidade foi implementada. Assim como a recuperação, o vídeo é individual.

Instruções para fazer o arquivo makefile. No [run.codes] tem uma orientação para que, no makefile, a diretiva "all" contenha apenas o comando para compilar seu programa e, na diretiva "run", apenas o comando para executá-lo. Assim, a forma mais simples de se fazer o arquivo makefile é:

```
all:
	gcc -o programaTrab *.c
run:
	./programaTrab
```

Lembrando que *.c já engloba todos os arquivos .c presentes no seu zip. Adicionalmente, no arquivo Makefile é importante se ter um *tab* nos locais colocados acima, senão ele pode não funcionar.

Instruções de entrega.

O programa deve ser submetido via [run.codes]:

- página: https://run.codes/Users/login
- código de matrícula: QLAT

O vídeo gravado deve ser submetido por meio de um formulário no qual o aluno vai informar o seu nome e o seu número USP, bem como um link do Google Drive que contém o vídeo gravado. Não deve ser usado o drive compartilhado da disciplina.

Critério de Correção

Critério de avaliação do trabalho. Na correção do trabalho, serão ponderados os seguintes aspectos.

- Corretude da execução do programa.
- Atendimento às especificações do registro de cabeçalho e dos registros de dados.
- Atendimento às especificações da sintaxe dos comandos de cada funcionalidade e do formato de saída da execução de cada funcionalidade.

- Qualidade da documentação entregue. A documentação interna terá um peso considerável no trabalho.
- Vídeo. Alunos que não apresentarem o vídeo receberão nota 0 na recuperação.

Casos de teste no [run.codes]. Juntamente com a especificação do trabalho, serão disponibilizados 50% dos casos de teste no [run.codes], para que os alunos possam avaliar o programa sendo desenvolvido. Os 50% restantes dos casos de teste serão utilizados nas correções.

Restrições adicionais sobre o critério de correção.

- A não execução de um programa devido a erros de compilação implica que a nota final da parte do trabalho será igual a zero (0).
- O não atendimento às especificações do registro de cabeçalho e dos registros de dados implica que haverá uma diminuição expressiva na nota do trabalho.
- O não atendimento às especificações de sintaxe dos comandos de cada funcionalidade e do formato de saída da execução de cada funcionalidade implica que haverá uma diminuição expressiva na nota do trabalho.
- A ausência da documentação implica que haverá uma diminuição expressiva na nota do trabalho.
- A inserção de palavras ofensivas nos arquivos e em qualquer outro material entregue implica que a nota final da parte do trabalho será igual a zero (0).
- Em caso de plágio, as notas dos trabalhos envolvidos serão zero (0).

Data de Entrega da Recuperação

Na data especificada na página da disciplina.

Bom Trabalho !