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

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

Aluno PAE
João Paulo Clarindo
jpcsantos@usp.br

Monitores
Eduardo Souza Rocha
eduardos.rocha17@usp.br ou telegram: @edwolt
Maria Júlia Soares De Grandi
maju.degrandi@usp.br ou *telegram: @majudegrandi*

Segundo Trabalho Prático

Este trabalho tem como objetivo indexar arquivos de dados usando um índice árvore-B.

O trabalho deve ser feito em duplas (ou seja, em 2 alunos). Os alunos devem ser os mesmos alunos do primeiro trabalho prático. Caso haja mudanças, elas devem ser informadas para a docente e os monitores. A solução deve ser proposta exclusivamente pelo(s) aluno(s) com base nos conhecimentos adquiridos nas aulas. Consulte as notas de aula e o livro texto quando necessário.

Fundamentos da disciplina de Bases de Dados

A disciplina de Estrutura de Dados III é uma disciplina fundamental para a disciplina de Bases de Dados. A definição deste primeiro trabalho prático é feita considerando esse aspecto, ou seja, o projeto é especificado em termos de várias funcionalidades, e essas funcionalidades são relacionadas com as funcionalidades da linguagem SQL (*Structured Query Language*), que é a linguagem utilizada por

sistemas gerenciadores de banco de dados (SGBDs) relacionais. As características e o detalhamento de SQL serão aprendidos na disciplina de Bases de Dados. Porém, por meio do desenvolvimento deste trabalho prático, os alunos poderão entrar em contato com alguns comandos da linguagem SQL e verificar como eles são implementados.

Este trabalho prático tem como objetivo implementar um arquivo de dados usando um índice árvore-B e, portanto, relacionar os conceitos da linguagem SQL com as funcionalidades providas pelo uso de índices.

Descrição de páginas de disco

No trabalho é usado o conceito de páginas de disco. Cada página de disco tem o tamanho fixo de 73 bytes. Como pode ser observado, o tamanho da página de disco adotado neste trabalho é lógico, sendo adaptado de acordo com os registros de dados do trabalho.

Importante 1. Na prática, o tamanho de página de disco do arquivo de dados e do arquivo de índice árvore-B é o mesmo. Entretanto, foi definido um tamanho diferente para o arquivo de índice árvore-B devido ao pequeno volume de dados indexado. Dessa forma, o arquivo já implementado no primeiro trabalho prático pode ser utilizado sem necessidade de alteração.

Importante 2. O tamanho da página de disco é sempre uma potência de 2. Neste trabalho foi adotado um tamanho diferente por simplificação.

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, pelo menos, da seguinte forma:

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

- Cada P_j $(1 \leq j \leq q)$ é um ponteiro para uma subárvore ou assume o valor -1 caso não exista subárvore.
- 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.

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 $1 < i < q$
   - $X < K_i$ para $i = 1$
   - $K_{i-1} < X$ 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*.

- *nroChavesTotal*: armazena o número de chaves de busca indexadas na árvore-B. Assume inicialmente o valor 0, sendo esse valor incrementado a cada inserção de uma nova chave – tamanho: inteiro de 4 *bytes*.
- *alturaArvore*: armazena a altura da árvore. Assume inicialmente o valor 0, sendo esse valor incrementado cada vez que a altura da árvore muda. Por exemplo, quando se tem a situação inicial correspondente a *nó-folha = nó raiz*, alturaArvore deve ser atualizada com o valor 1. Depois do primeiro *split*, alturaArvore deve ser atualizada com o valor 2. E assim sucessivamente.
- *RRNproxNo*: armazena o RRN do próximo nó (página) do índice árvore-B a ser inserido. Assume inicialmente o valor 0, sendo esse valor incrementado a cada criação de um novo nó – tamanho: inteiro de 4 *bytes*.

Representação Gráfica do Registro de Cabeçalho. O tamanho do registro de cabeçalho deve ser de 73 bytes, representado da seguinte forma:

1 byte	4 bytes				4 bytes				4 bytes				4 bytes				56 bytes
status	*noRaiz*				nroChavesTotal				*RRNproxNo*				*RRNproxNo*				*lixo identificado pelo caractere '$'*
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17…72

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.
- Os bytes restantes para o preenchimento do tamanho da página de disco devem ser preenchidos com lixo. O lixo é representado pelo caractere '$'. Nenhum *byte* do registro de cabeçalho deve permanecer vazio, ou seja, cada *byte* deve armazenar um valor válido ou '$'.

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:

- *folha*: indica se o nó é um nó folha ou não. Pode assumir os valores 0, para indicar que o nó não é folha, ou 1, para indicar que o nó é folha – tamanho: *string* de 1 byte.
- *nroChavesNo*: armazena o número de chaves de busca indexadas no nó – tamanho: inteiro de 4 *bytes*.
- *alturaNo*: armazena a altura do nó na árvore. Quando o nó é um nó-folha, *alturaNo* deve ser igual a 1. Quando o nó é pai de um nó-folha, então *alturaNo* deve ser igual a 2. E assim sucessivamente. - tamanho: inteiro de 4 *bytes*.
- *RRNdoNo*: armazena o número do RRN referente ao nó (ou seja, à página de disco) - tamanho: inteiro de 4 *bytes*.

Detalhes sobre o índice árvore-B

A ordem da árvore-B é 5, ou seja, ***m = 5***. Portanto, um nó (página) terá 4 chaves de busca e 5 descendentes. Lembrando que, em aplicações reais, a ordem da árvore-B é muito maior, para acomodar mais chaves de busca. A proposta da árvore-B é que ela seja larga e baixa, para diminuir o número de acessos a disco. Adicionalmente, a **chave de busca** é o **campo *idConecta***. Esse é, portanto, o campo a ser indexado.

Detalhes sobre o algoritmo de inserção. Considere que deve ser implementada a rotina de *split* durante a inserção. Considere que a rotina de redistribuição durante a inserção não deve ser implementada. Considere que a distribuição das chaves de busca deve ser o mais uniforme possível, ou seja, considere que a chave de busca a ser promovida deve ser a chave que distribui o mais uniformemente possível as demais chaves entre o nó à esquerda e o novo nó resultante do particionamento. Em situações nas quais a ordem é ímpar, a chave de busca a ser promovida é a chave que distribui uniformemente as demais chaves entre o nó à esquerda e o novo nó resultante do particionamento. Em situações nas quais a ordem é par, 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 deve 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 de cada registro de dados é de 73 bytes, representado da seguinte forma:

1 byte	4 bytes	4 bytes	4 bytes	4 bytes	4 bytes	4 bytes	4 bytes	4 bytes	4 bytes
folha	*nroChavesNo*	*alturaNo*	*RRNdoNo*	P_1	C_1	P_{R1}	P_2	C_2	P_{R2}
0	1...4	5...8	9...12	13...16	17...20	21...24	25...28	29...32	33...36

4 bytes	4 bytes	4 bytes	4 bytes	4 bytes	4 bytes	4 bytes	4 bytes	4 bytes
C_2	P_{R2}	P_3	C3	P_{R3}	P_4	C_4	P_{R4}	P_5
37...40	41...44	45...48	49...52	53...56	57...60	61...64	65...86	69...72

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. O valor -1 também deve ser usado para denotar ponteiros P_i $(1 \leq i \leq m)$ e P_{Ri} $(1 \leq i \leq m-1)$ que sejam nulos.

Programa

Descrição Geral. Implemente um programa em C por meio do qual seja possível realizar a pesquisa e a inserção de dados em arquivos de dados com o auxílio de um índice árvore-B.

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:

Na linguagem SQL, o comando CREATE TABLE é usado para criar uma tabela, a qual é implementada como um arquivo. Geralmente, indica-se um campo (ou um conjunto de campos) que consiste na chave primária da tabela. Isso é realizado especificando-se a cláusula PRIMARY KEY. A funcionalidade [7] representa um exemplo de implementação do índice árvore-B decorrente da especificação da cláusula PRIMARY KEY.

Na linguagem SQL, o comando CREATE INDEX é usado para criar um índice sobre um campo (ou um conjunto de campos) que consiste na chave primária de uma tabela já existente. A funcionalidade [7] representa um exemplo de implementação do índice árvore-B.

[7] Crie um arquivo de índice árvore-B que indexa a chave de busca definida sobre um arquivo de dados de entrada já existente, o qual foi definido no primeiro trabalho prático e pode conter registros logicamente removidos. Entretanto, 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. Considere que não existe repetição nos valores de busca e que, portanto, essa situação não precisa ser tratada. 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 [7]:
`7 arquivoDados.bin arquivoIndice.bin`
onde:
- `arquivoDados.bin` é um arquivo binário de entrada que segue as mesmas especificações do arquivo de dados 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 a chave de busca. 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
7 topologiaRede.bin indiceTopologiaRede.bin
```
`usar a função binarioNaTela antes de terminar a execução da funcionalidade, para mostrar a saída do arquivo indiceTopologiaRede.bin, o qual indexa a chave de busca definida neste trabalho prático.`

Uma vez criados os índices necessários, eles são automaticamente utilizados pelo SGBD para atender aos demais comandos solicitados, de forma transparente.

Conforme visto na funcionalidade [2], na linguagem SQL o comando SELECT é usado para listar os dados de uma tabela. Existem várias cláusulas que compõem o comando SELECT. Além das cláusulas SELECT e FROM, outra cláusula muito comum é a cláusula WHERE, que permite que seja definido um critério de busca sobre um ou mais campos, o qual é nomeado como critério de seleção.

SELECT lista de colunas (ou seja, campos a serem exibidos na resposta)
FROM tabela (ou seja, arquivo que contém os campos)
WHERE critério de seleção (ou seja, critério de busca)

A funcionalidade [8] representa um exemplo de implementação do comando SELECT considerando a cláusula WHERE. Como existe um índice árvore-B definido sobre a chave de busca, a implementação dessa funcionalidade consiste em utilizar esse índice para recuperar os dados desejados.

[8] Permita a recuperação dos dados de todos os registros de um arquivo de dados de entrada, de forma que esses registros satisfaçam um critério de busca determinado pelo usuário. Qualquer campo pode ser utilizado como forma de busca. A semântica, as características e a saída da funcionalidade [8] são as mesmas que as da funcionalidade [3], com a seguinte diferença. Quando o campo usado na busca for a chave de busca, então a busca deve ser feita usando o índice árvore-B criado na funcionalidade [7]. Caso contrário, ou seja, quando o campo usado na busca não for a chave de busca, então a busca deve ser feita usando a funcionalidade [3]. A manipulação do arquivo de índice árvore-B deve ser feita em disco, de acordo com o conteúdo ministrado em sala de aula.

Sintaxe do comando para a funcionalidade [8]:

```
8 arquivoDados.bin arquivoIndice.bin n
nomeCampo1 valorCampo1
nomeCampo2 valorCampo2
...
nomeCampon valorCampon
```

onde:

- `arquivoDados.bin` é um arquivo binário de entrada que segue as mesmas especificações do arquivo de dados do primeiro trabalho prático.

- `arquivoIndice.bin` é o arquivo binário de índice árvore-B que indexa a chave de busca. Esse arquivo deve seguir as especificações definidas neste trabalho prático.

- `n` é a quantidade de vezes que a busca deve ser realizada. Para cada busca, deve ser indicado o nome do Campo e o valor do Campo utilizados na busca. Cada um dos n (nomeCampo valorCampo) deve ser especificado em uma linha diferente. Deve ser deixado um espaço em branco entre nomeCampo e valorCampo. Os valores dos campos do tipo *string* devem ser especificados entre aspas duplas (").

Saída caso o programa seja executado com sucesso:

Para cada valor de n, exiba a seguinte saída. Cada campo de cada registro deve ser mostrado em uma única linha contendo "descrição detalhada do campo: valor do campo". Caso o campo seja nulo, então não devem ser mostrados sua descrição detalhada e seu valor. Quanto à velocidade de transmissão, deve ser colocado o valor da velocidade e o valor da medida de velocidade, o qual deve ser concatenado com a *string* bps, gerando Mbps ou Gbps. Deixe uma linha em branco depois que todos os campos de um registro forem exibidos. A ordem de exibição dos campos bem como os campos que devem ser exibidos podem ser vistos no **exemplo de execução.** No final, mostre o número de páginas de disco acessadas na busca. Caso a busca seja feita usando o campo correspondente à chave de busca, então o número de páginas de disco deve ser calculado da seguinte forma: página de disco do registro de cabeçalho da árvore-B + número de páginas de disco acessadas na manipulação do arquivo da árvore-B + número de páginas de disco acessadas na manipulação do arquivo de dados, incluindo a manipulação do registro de cabeçalho desse arquivo de dados. Caso contrário, ou seja, caso a busca não seja feita usando o campo correspondente à chave de busca, o número de páginas de disco deve ser calculado da seguinte forma: página de disco referente ao registro de cabeçalho e aos registros de dados.

Mensagem de saída caso não seja encontrado o registro ou que o registro esteja removido:

```
Registro inexistente.
```

Mensagem de saída caso algum erro seja encontrado:

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

Exemplo de execução:

```
./programaTrab
8 topologiaRede.bin indiceTopologiaRede.bin 2
nomePoPs "Recife" --->Essa busca é realizada pela funcionalidade [3]
idConecta 23      --->Essa busca é realizada usando a árvore-B
Saída
Busca 1
Identificador do ponto: 1
Nome do ponto: Recife
Pais de localizacao: Brazil
Sigla do pais: BR
Identificador do ponto conectado: 22
Velocidade de transmissao: 3 Mbps
**** linha em branco ****
Numero de paginas de disco: 55
**** linha em branco ****
Busca 2
Identificador do ponto: 23
Nome do ponto: Natal
Pais de localizacao: Brazil
Sigla do pais: BR
Identificador do ponto conectado: 31
Velocidade de transmissao: 10 Gbps
**** linha em branco ****
Numero de paginas de disco: 4
**** linha em branco ****
```

Na linguagem SQL, o comando INSERT INTO é usado para inserir dados em uma tabela. Para tanto, devem ser especificados os valores a serem armazenados em cada coluna da tabela, de acordo com o tipo de dado definido. A funcionalidade [9] representa um exemplo de implementação do comando INSERT INTO.

[9] Permita a inserção de novos registros em um arquivo de dados de entrada, de acordo com a especificação da funcionalidade [5]. Na implementação da funcionalidade [9], depois da inserção do registro no arquivo de dados de entrada, deve ser realizada a inserção de sua chave de busca correspondente 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 dos arquivos binários.

Entrada do programa para a funcionalidade [9]:

`9 arquivoDados.bin arquivoIndice.bin n`
`idConecta`$_1$ `nomePoPs`$_1$ `nomePais`$_1$ `siglaPais`$_1$ `idPoPsConectado`$_1$ `medidaVelocidade`$_1$ `velocidade`$_1$
`idConecta`$_2$ `nomePoPs`$_2$ `nomePais`$_2$ `siglaPais`$_2$ `idPoPsConectado`$_2$ `medidaVelocidade`$_2$ `velocidade`$_2$
`...`
`idConecta`$_n$ `nomePoPs`$_n$ `nomePais`$_n$ `siglaPais`$_n$ `idPoPsConectado`$_n$ `medidaVelocidade`$_n$ `velocidade`$_n$

onde:

- `arquivoDados.bin` é o arquivo binário de entrada, o qual foi gerado conforme as especificações descritas 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 a chave de busca. 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: *idConecta*, *nomePoPs*, *nomePais*, *siglaPais*, *idPoPsConectado*, *medidaVelocidade*, *velocidade*. 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 os arquivos no formato binário usando a função fornecida binarioNaTela.

Mensagem de saída caso algum erro seja encontrado:

Falha no processamento do arquivo.

Exemplo de execução:

```
./programaTrab
9 topologiaRede.bin indiceTopologiaRede.bin 2
3 "Campina Grande" "Brazil" "BR" 4 "G" 10
22 "Teresina" NULO "BR" NULO NULO NULO
```

usar a função binarioNaTela antes de terminar a execução da funcionalidade, para mostrar a saída dos arquivos topologiaRede.bin e indiceTopologiaRede.bin, os quais foram atualizados frente às inserções.

Na linguagem SQL, o comando SELECT é usado para listar os dados de uma ou mais tabelas (arquivos de dados). Quando mais do que uma tabela é usada, essas tabelas são "juntadas" por meio de uma (ou mais) coluna(s) de junção. Existem duas diferentes formas de se especificar a junção no comando SELECT:

SELECT lista de colunas (ou seja, campos a serem exibidos na resposta)
FROM tabela1, tabela2 (ou seja, arquivos que contêm os campos)
(podem existir mais tabelas)
WHERE tabela1.nomeCampo = tabela2.nomeCampo (condição de junção)
AND critério de seleção (ou seja, critério de busca)

ou

SELECT lista de colunas (ou seja, campos a serem exibidos na resposta)
FROM tabela1 JOIN tabela2 ON tabela1.nomeCampo = tabela2.nomeCampo
(ou seja, arquivos que contêm os campos + condição de junção)
(podem existir mais tabelas)
WHERE critério de seleção (ou seja, critério de busca)

A operação de junção é amplamente utilizada em aplicações de banco de dados desde que ela "junta" dados de registros de dois arquivos usando como base um campo de igualdade (condição de junção).

Considere um primeiro arquivo de dados arquivoA com os campos Acampo1, Acampo2, …, AcampoN. Considere um segundo arquivo de dados arquivoB com os campos Bcampo1, Bcampo2, …, BcampoM. As seguintes situações podem ocorrer na junção de arquivoA e arquivoB considerando a condição de junção Acampo1 = Bcampo1:

(1) Não existe igualdade entre Acampo1 e Bcampo1. Nesse caso, nenhum registro é gerado como resultado da junção.
(2) Existe apenas uma igualdade entre Acampo1 e Bcampo1. Nesse caso, somente um registro é gerado como resultado da junção.
(3) Existem k igualdades entre Acampo1 e Bcampo1. Nesse caso, são gerados k registros como resultado da junção.

[10] Permita a recuperação dos dados de todos os registros armazenados no arquivo de dados topologiaRede.bin considerando o campo *idConecta*, juntando-os de forma apropriada com os dados de topologiaRede.bin considerando o campo *idPoPsConectado*. Essa funcionalidade requer a realização de uma autojunção, ou seja, o mesmo arquivo é usado duas vezes na junção, considerando como condição de junção **topologiaRede1.idConecta = topologiaRede2.idPoPsConectado**. Existem várias formas de se implementar a junção e, nesta funcionalidade, ela deve ser implementada por meio da **junção de loop único**. A junção de loop único depende da existência de índices. Portanto, o índice criado neste trabalho prático deve ser usado.

O algoritmo da **junção de loop único** é definido como segue. Para cada registro presente no arquivo `arquivoA` (loop externo), recupere cada registro do arquivo `arquivoB` (loop interno) usando o índice e teste se os dois registros satisfazem à condição de junção `Acampo1 = Bcampo1`. Ou seja:

para cada registro em `topologiaRede1` faça
    selecione os registros de `topologiaRede2` que satisfaçam à condição
        `topologiaRede1.idPoPsConectado= topologiaRede2.idConecta`
        usando o índice árvore-B definido sobre o arquivo `topologiaRede2`
    mostre a resposta conforme solicitado
fim-para

Entrada do programa para a funcionalidade [10]:
`10 arquivoDados1.bin arquivoDados2.bin nomeCampoArquivo1, nomeCampoArquivo2, arquivoIndiceDados2.bin`
onde:
- `arquivoDados1.bin` é um arquivo binário de entrada que segue as mesmas especificações do arquivo de dados definido no primeiro trabalho prático, e que contém dados desordenados e registros logicamente removidos.
- `arquivoDados2.bin` é um arquivo binário de entrada que segue as mesmas especificações do arquivo de dados definido no primeiro trabalho prático, e que contém dados desordenados e registros logicamente removidos.
- `nomeCampoArquivo1` é o nome do campo do arquivo de dados `arquivoDados1.bin` que está sendo usado como condição de junção. Neste trabalho prático, apenas o campo **idPoPsConectado** pode ser utilizado.
- `nomeCampo Arquivo2` é o nome do campo do arquivo de dados `arquivoDados1.bin` que está sendo usado como condição de junção. Neste trabalho prático, apenas o campo **idConecta** pode ser utilizado.
- `indiceLinha` é o índice árvore-B definido sobre o arquivo de dados `arquivoIndiceDados2`.

Mensagem de saída caso algum erro seja encontrado:
`Falha no processamento do arquivo.`

Mensagem de saída caso não seja gerado nenhum registro na junção dos dois arquivos:
`Registro inexistente.`

Saída caso o programa seja executado com sucesso:
`A saída deve ser exibida da seguinte forma. Para cada registro do arquivo topologiaRede1.bin, primeiro mostre os seus campos. Depois, mostre os campos do registro do arquivo topologiaRede2.bin correspondente. Os campos a serem exibidos seguem as mesmas regras definidas para a funcionalidade [3]. Depois de exibidos todos os campos, pule uma linha em branco. A ordem de exibição dos campos dos registros é ilustrada no` **`exemplo de execução.`**

Exemplo de execução:
```
./programaTrab
10 topologiaRede1.bin topologiaRede2.bin idPoPsConectado idConecta indiceTopologiaRede2.bin

Identificador do ponto: 23 ---> valor de idConecta em topologiaRede1
Nome do ponto: Natal ---> valor de nomePoPs em topologiaRede1
Pais de localizacao: Brazil ---> valor de nomePais em topologiaRede1
Sigla do pais: BR ---> valor de siglaPais em topologiaRede1
```

```
Identificador do ponto conectado: 31 ---> valor de idPoPsConectado em
 topologiaRede1
Nome do ponto conectado: Fortaleza ---> valor de nomePoPs em topologiaRede2
Sigla do pais conectado: Brazil ---> valor de nomePais em topologiaRede2
Sigla do pais: BR ---> valor de siglaPais em topologiaRede2
Velocidade de transmissao: 10 Gbps ---> valor de velocidade e de unidadeMedida
em topologiaRede1
**** linha em branco ****
Identificador do ponto: 50 ---> valor de idConecta em topologiaRede1
Nome do ponto: Osono ---> valor de nomePoPs em topologiaRede1
Pais de localizacao: Chile ---> valor de nomePais em topologiaRede1
Sigla do pais: CL ---> valor de siglaPais em topologiaRede1
Identificador do ponto conectado: 55 ---> valor de idPoPsConectado em
 topologiaRede1
Nome do ponto conectado: Valdivia ---> valor de nomePoPs em topologiaRede2
Sigla do pais conectado: Chile ---> valor de nomePais em topologiaRede2
Sigla do pais: CL ---> valor de siglaPais em topologiaRede2
Velocidade de transmissao: 155 Mbps ---> valor de velocidade e de
unidadeMedida em topologiaRede1
**** linha em branco ****
```

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.

[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 no 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.
- Uma especificação da participação de cada integrante no desenvolvimento do trabalho. Isso deve ser feito logo no início do arquivo que contém a função **main**. Deve ser indicado, para cada integrante, seu número USP, seu nome completo e sua porcentagem de participação. A porcentagem de participação deve variar entre 0% e 100%. Por exemplo, o integrante desenvolveu 100% do

que era esperado de sua parte, ou então o integrante desenvolveu 80% do que era esperado para a sua parte.

- Um vídeo gravado pelos integrantes do grupo, o qual deve ter, no máximo, 5 minutos de gravação. O vídeo deve explicar o trabalho desenvolvido. Ou seja, o grupo deve apresentar: cada funcionalidade e uma breve descrição de como a funcionalidade foi implementada. Todos os integrantes do grupo devem participar do vídeo, sendo que o tempo de apresentação dos integrantes deve ser balanceado. Ou seja, o tempo de participação de cada integrante deve ser aproximadamente o mesmo. O uso da webcam é obrigatório.

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: **1YGS**

O vídeo gravado deve ser submetido por meio da página da disciplina no e-disciplinas, no qual o grupo vai informar o nome de cada integrante, o número do grupo e um link

que contém o vídeo gravado. Ao submeter o link, verifique se o mesmo pode ser acessado. Vídeos cujos links não puderem ser acessados receberão nota zero.

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. Integrantes que não participarem da apresentação receberão nota 0 no trabalho correspondente.

Casos de teste no [run.codes]. Juntamente com a especificação do trabalho, serão disponibilizados 70% dos casos de teste no [run.codes], para que os alunos possam avaliar o programa sendo desenvolvido. Os 30% 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 realização do trabalho prático com alunos de turmas diferentes 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 do Trabalho

Na data especificada na página da disciplina.
