Universidade de São Paulo
Instituto de Ciências Matemáticas e de Computação
Departamento de Ciências de Computação
Disciplina de Algoritmos e Estruturas de Dados II
docente
Profa. Dra. Cristina Dutra de Aguiar Ciferri (cdac@icmc.usp.br)
aluno PAE
Victor Hugo Andrade Soares (victorhugoasoares@usp.br)
monitor
João Vitor dos Santos Tristão (joaovitortristao@usp.br)

Quarto Trabalho Prático

Este trabalho tem como objetivo indexar arquivos de dados usando um índice simples ou linear.

O trabalho deve ser feito em ***dupla*** ou ***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. A dupla deve escolher um dos trabalhos anteriores a ser utilizado para desenvolver o quarto trabalho prático. Isso deve ser informado no início do código como um comentário. Os nomes dos integrantes da dupla também devem ser definidos no início do código.*

Descrição de páginas de disco

No trabalho será usado o conceito de páginas de disco. Cada página de disco tem o tamanho fixo de 16.000 bytes. O conceito de página de disco é um conceito lógico, ou seja, deve ser garantido via programação, de forma que cada página de disco contenha, no máximo, o tamanho fixo especificado.

Descrição do arquivo de índice simples ou linear

Descrição do Registro de Cabeçalho. O registro de cabeçalho deve conter o seguinte campo:

- *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.
- *nroRegistros*: indica a quantidade de registros do arquivo de índice (sem contar o registro de cabeçalho, ou seja, somente os registros de dados do arquivo de índice) – tamanho: inteiro de 4 bytes.

Representação Gráfica do Registro de Cabeçalho. O tamanho do registro de cabeçalho deve ser de 5 byte, representado da seguinte forma:

1 byte	4 bytes
status	*nroRegistro*

Página de disco. O registro de cabeçalho deve ocupar uma página de disco. Seu tamanho é menor do que o tamanho da página de disco. Neste caso, a página de disco deve ser preenchida com caractere '@' até completar o seu tamanho.

Descrição do Registro de Dados. Devem ser considerados campos de tamanho fixo e registros de tamanho fixo. Os registros de dados devem conter os seguintes campos:

- chaveBusca: armazena a chave de busca do índice - tamanho: *string* de 28 bytes.
- RRN: armazena o RRN do registro que corresponde à chave de busca associada – tamanho: inteiro de 4 bytes.

Representação Gráfica do Registro de Dados. O tamanho de cada registro de dados é de 32 bytes, representado da seguinte forma:

28 bytes			4 bytes
chaveBusca			*RRN*
0	1	.....	31

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.

Programa

Descrição Geral. Implemente um programa em C que ofereça uma interface por meio da qual o usuário possa realizar operações sobre um índice simples ou linear e que possa inserir, remover e atualizar dados de um arquivo de dados que está indexado por esse índice simples ou linear.

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 "programaTrab4". Essas orientações devem ser seguidas uma vez que a correção do funcionamento do programa se dará de forma automática.

Descrição Específica. O programa deve oferecer as seguintes funcionalidades:

[11] Crie um arquivo de índice secundário fortemente ligado para um arquivo de dados de entrada já existente. O campo a ser indexado é *nomeEscola*. A implementação do índice não deve ser na forma de lista invertida, ou seja, os valores repetidos de *nomeEscola* devem aparecer várias vezes no arquivo de dados do índice. Registros logicamente removidos presentes no arquivo de dados de entrada não devem ter suas chaves de busca correspondentes no arquivo de índice. Registros com nomes de escola nulos também não devem ter suas chaves de busca correspondentes no arquivo de índice. No arquivo de índice, as chaves de busca devem seguir a ordenação alfabética, desde que as chaves de busca são do tipo *string*. Adicionalmente, quando duas ou mais chaves de busca possuírem o mesmo valor para o campo *nomeEscola*, os valores dos *RRNs* correspondentes devem ser ordenados de forma crescente. A ordenação deve ser numérica, desde que os *RRNs* são do tipo inteiro. A implementação dessa funcionalidade deve ser feita de forma eficiente. Antes de terminar a execução da funcionalidade, deve ser utilizada a função binarioNaTela1 ou binarioNaTela2, ambas disponibilizadas na página do projeto da disciplina, para mostrar a saída do arquivo de índice secundário.

Entrada do programa para a funcionalidade [11]:
`11 arquivoEntrada.bin arquivoIndiceNomeEscola.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.
- `arquivoIndiceNomeEscola.bin` é o arquivo binário de índice secundário fortemente ligado que indexa o campo *nomeEscola*. Esse arquivo deve seguir as especificações definidas neste trabalho prático.

Saída caso o programa seja executado com sucesso:
`Listar o arquivo binário arquivoIndiceNomeEscola.bin.`
Mensagem de saída caso algum erro seja encontrado:
`Falha no processamento do arquivo.`
Exemplo de execução:
```
./programaTrab4
11 arquivo.bin arquivoIndiceNomeEscola.bin
usar a função binarioNaTela1 ou binarioNaTela2 antes de terminar a execução da funcionalidade, para mostrar a saída do arquivo arquivoIndiceNomeEscola.bin, o qual indexa o campo nomeEscola.
```

[12] Permita a recuperação dos dados de todos os registros que satisfaçam um critério de busca determinado pelo usuário sobre o campo *nomeEscola*, usando o índice secundário fortemente ligado criado na funcionalidade [11]. Note que somente o campo *nomeEscola* deve ser utilizado como forma de busca, desde que o índice criado na funcionalidade [11] indexa chaves de busca desse campo. Essa funcionalidade pode retornar 0 registros (quando nenhum satisfaz ao critério de busca), 1 registro (quando não existem valores repetidos do nome do servidor) ou vários registros (quando existem valores repetidos do nome do servidor). Antes de executar essa funcionalidade, realize a operação de carregamento do índice secundário do disco para a memória primária. Os dados solicitados devem ser mostrados na saída padrão da seguinte forma. Para cada registro, mostre os metadados, seguidos por dois pontos (:), seguidos de um espaço em branco, seguidos pelos valores de seus campos, de forma que cada campo apareça em uma linha diferente. Os metadados devem ser lidos do registro de cabeçalho (campos *desCampo1*, *desCampo2*, *desCampo3*, *desCampo4*, *desCampo5*). Caso o valor seja nulo, escreva 'valor nao declarado'. Depois do registro, deve-se pular uma linha em branco. Depois de mostrar todos os registros, deve ser mostrado na saída padrão o número de páginas de disco acessadas, especificadas da seguinte forma: (i) número de acessos a disco na operação de carregamento do índice secundário (contabilizar o número de acessos ao registro de cabeçalho e aos registros de dados); e (ii) número de acessos a disco para acessar o arquivo de dados.

Sintaxe do comando para a funcionalidade [12]:

```
12 arquivo.bin arquivoIndiceNomeEscola.bin nomeEscola valor
```

Saída caso o programa seja executado com sucesso:

Podem ser encontrados vários registros que satisfaçam à condição de busca. Cada registro deve ser mostrado em uma única linha e os seus campos devem ser mostrados de forma sequencial separado por espaço. Campos de tamanho fixo que tiverem o valor nulo não devem mostrados. Para os campos com tamanho variável, mostre também sua quantidade de caracteres (tamanhos em bytes sem o caractere '\0'). Para os campos de tamanho variável com valores nulos, não deve ser exibido nada. Especifique também o número de páginas de disco para carregar o arquivo de índice e o número de páginas de disco para acessar o arquivo de dados.

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 apenas o primeiro registro que satisfez à busca, embora a funcionalidade provida pelo programa deva exibir mais do que um registro quando for o caso)**:**

```
./programaTrab4
12 arquivo.bin arquivoIndiceNomeEscola.bin nomeEscola BARROS CONEGO
2104 457 13 BARROS CONEGO
Número de páginas de disco para carregar o arquivo de índice: 11
Número de páginas de disco para acessar o arquivo de dados: 1
```

[13] Estenda a funcionalidade [5] descrita no segundo trabalho prático de forma que, depois de cada remoção lógica no arquivo de dados, a chave de busca referente ao registro logicamente removido seja removida do índice secundário fortemente ligado criado na funcionalidade [11]. A especificação detalhada da funcionalidade [13] é a mesma da funcionalidade [5]. Com relação à manipulação do índice secundário, antes de executar a funcionalidade [13], realize a operação de carregamento do índice do disco para a memória primária. Enquanto forem realizadas as *n* remoções, o índice deve ser atualizado somente em memória primária. Ao finalizar as *n* remoções, realize a operação de reescrita do índice da memória primária para o disco. Nessa operação de reescrita, use a opção *wb* para reescrever no arquivo.

Entrada do programa para a funcionalidade [13]:

`13 arquivo.bin arquivoIndiceNomeEscola.bin n`
`nomeCampo`$_1$ `valorCampo`$_1$
`nomeCampo`$_2$ `valorCampo`$_2$
`...`
`nomeCampo`$_n$ `valorCampo`$_n$

onde:

- `arquivo.bin` é um arquivo binário de entrada que segue as mesmas especificações definidas no primeiro trabalho prático. As remoções a serem realizadas nessa funcionalidade devem ser feitas nesse arquivo.
- `arquivoIndiceNomeEscola.bin` é o arquivo binário referente ao índice secundário fortemente ligado que segue as especificações definidas nesse trabalho prático.
- `n` é o número de remoções a serem realizadas, as quais seguem as mesmas especificações definidas na funcionalidade [5] do segundo trabalho prático. Ou seja, para cada remoção, deve ser informado o `nome do campo` a ser considerado e seu critério de busca representado pelo `valor do campo`. Cada uma das `n` remoções deve ser especificada em uma linha diferente. Deve ser deixado um espaço em branco entre o `nome do campo` e o `valor do campo`. Os valores dos campos do tipo *string* devem ser especificados entre aspas duplas (").

Saída caso o programa seja executado com sucesso:

`Listar o arquivo de índice secundário arquivoIndiceNomeEscola.bin.`

Mensagem de saída caso algum erro seja encontrado:

`Falha no processamento do arquivo.`

Exemplo de execução:

```
./programaTrab4
13 arquivo.bin arquivoIndiceNomeEscola.bin 2
nroInscricao 546
data "14/01/2004"
usar a função binarioNaTela1 ou binarioNaTela2 antes de terminar a execução da funcionalidade, para mostrar a saída do arquivo arquivoIndiceNomeEscola.bin, o qual foi atualizado frente às remoções realizadas no arquivo de dados.
```

[14] Estenda a funcionalidade [6] descrita no segundo trabalho prático de forma que, depois de cada inserção de registro adicional no arquivo de dados, a chave de busca referente ao registro inserido seja inserida no índice secundário fortemente ligado criado na funcionalidade [11]. A especificação detalhada da funcionalidade [14] é a mesma da funcionalidade [6]. Com relação à manipulação do índice secundário, antes de executar a funcionalidade [14], realize a operação de carregamento do índice do disco para a memória primária. Enquanto forem realizadas as *n* inserções, o índice deve ser atualizado somente em memória primária. Ao finalizar as *n* inserções, realize a operação de reescrita do índice da memória primária para o disco. Nessa operação de reescrita, use a opção *wb* para reescrever no arquivo.

Entrada do programa para a funcionalidade [14]:

`14 arquivo.bin arquivoIndiceNomeEscola.bin n`
`valorNroInscricao`$_1$ `valorNota`$_1$ `valorData`$_1$ `valorCidade`$_1$ `valorNomeEscola`$_1$
`valorNroInscricao`$_2$ `valorNota`$_2$ `valorData`$_2$ `valorCidade`$_2$ `valorNomeEscola`$_2$
`...`
`valorNroInscricao`$_n$ `valorNota`$_n$ `valorData`$_n$ `valorCidade`$_n$ `valorNomeEscola`$_n$

onde:

- `arquivo.bin` é um arquivo binário de entrada que segue as mesmas especificações definidas no primeiro trabalho prático. As inserções a serem realizadas nessa funcionalidade devem ser feitas nesse arquivo.
- `arquivoIndiceNomeServidor.bin` é o arquivo binário referente ao índice secundário fortemente ligado que segue as especificações definidas nesse trabalho prático.
- `n` é o número de inserções a serem realizadas, as quais seguem as mesmas especificações definidas na funcionalidade [6] do segundo trabalho prático. Ou seja, para cada inserção, deve ser informado os valores a serem inseridos no arquivo de dados, para os campos especificados na mesma ordem que a definida no primeiro trabalho prático, a saber: `nroInscrição`, `nota`, `data`, `cidade`, `nomeEscola`. Não existe truncamento de dados. 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 de índice secundário arquivoIndiceNomeEscola.bin.`

Mensagem de saída caso algum erro seja encontrado:

`Falha no processamento do arquivo.`

Exemplo de execução:

```
./programaTrab4
14 arquivo.bin arquivoIndiceNomeEscola.bin 2
1234 109.98 NULO NULO "ESCOLA DE ESTUDO PRIMÁRIO"
2132 408.02 "01/08/2016" "CAMPINAS" NULO
usar a função binarioNaTela1 ou binarioNaTela2 antes de terminar a execução da funcionalidade, para mostrar a saída do arquivo arquivoIndiceNomeEscola.bin, o qual foi atualizado frente às inserções realizadas no arquivo de dados.
```

[15] Permita a realização de estatísticas considerando a recuperação dos dados de todos os registros que satisfaçam um critério de busca determinado pelo usuário sobre o campo *nomeEscola*. Essa funcionalidade deve ser implementada da seguinte forma. Dado o nome de uma escola como entrada, primeiramente execute a funcionalidade [3] definida no primeiro trabalho prático. Em seguida, execute a funcionalidade [12] definida no presente trabalho prático. Ao final, liste a diferença no número de páginas de disco acessadas, contabilizadas da seguinte forma: número de páginas de disco acessadas na funcionalidade [3] *menos* o número de páginas de disco para acessar o arquivo de dados na funcionalidade [12]. Note que o número de páginas de disco para carregar o arquivo de índice da funcionalidade [12] foi desconsiderado no cálculo porque, em uma aplicação real, o carregamento do índice é feito apenas uma única vez no início da execução do programa, sendo o índice reescrito em disco somente no final da execução do programa. Antes de exibir o resultado da funcionalidade [3], exiba a frase "*** Realizando a busca sem o auxílio de índice". Depois exiba o resultado da funcionalidade [3]. Deixe uma linha em branco. Adicionalmente, antes de exibir o resultado da funcionalidade [12], exiba a frase "*** Realizando a busca com o auxílio de um índice secundário fortemente ligado". Depois, exiba o resultado da funcionalidade [12]. Deixe uma linha em branco. Na sequencia, exiba a diferença no número de páginas de disco acessadas.

Sintaxe do comando para a funcionalidade [15]:

```
15 arquivo.bin arquivoIndiceNomeEscola.bin nomeEscola valor
```

Saída caso o programa seja executado com sucesso:

```
Liste na sequencia descrita a seguir:
1- A frase: *** Realizando a busca sem o auxílio de índice
2- O resultado da execução da funcionalidade [3], conforme
especificado no segundo trabalho prático
3- Pular uma linha em branco
4- A frase: *** Realizando a busca com o auxílio de um índice
secundário fortemente ligado
5- O resultado da execução da funcionalidade [12], conforme
especificado neste trabalho prático
6- Pular uma linha em branco
7- A frase: Diferença no número de páginas de disco acessadas:
8- O resultado da diferença, dado pelo número de páginas de disco
acessadas na funcionalidade [3] menos o número de páginas de disco
para acessar o arquivo de dados na funcionalidade [12].
```

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 (considere que os metadados lidos do arquivo .csv sejam: numero de identificacao do servidor, salario do servidor, telefone celular do servidor, nome do servidor, cargo do servidor):

```
./programaTrab4
15 arquivo.bin arquivoIndiceNomeEscola.bin nomeEscola BARROS CONEGO
*** Realizando a busca sem o auxílio de índice
2104 457 13 BARROS CONEGO
Número de páginas de disco acessadas: 26
--- pular uma linha em branco ----
*** Realizando a busca com o auxílio de um índice secundário
fortemente ligado
2104 457 13 BARROS CONEGO
Número de páginas de disco para carregar o arquivo de índice: 10
Número de páginas de disco para acessar o arquivo de dados: 1
--- pular uma linha em branco ----
Diferença no número de páginas de disco acessadas: 25
```

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 e lidos campo a campo. Ou seja, não é possível escrever e ler os dados registro a 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. 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.

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 programaTrab2 *.c
run:
./programaTrab2
```

Lembrando que *.c já engloba todos os arquivos .c presentes no seu zip."

Instruções de entrega. A entrega deve ser feita via [run.codes]:

- página: https://run.codes/Users/login
- código de matrícula para a **Turma A**: **SRAZ**
- código de matrícula para a **Turma B**: **29QF**

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.

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 do Trabalho

Na data especificada na página da disciplina.

Bom Trabalho !