* Acesso Sequencial Indexado

1. Definição de itens por página

Com a intenção de otimizar o método em geral, optamos por dinamizar o tamanho das páginas de acordo com o tamanho da entrada. Como o número de registros está bem definido no enunciado do trabalho, não nos preocupamos com grandezas diferentes, haja vista que a verificação de parâmetros ao início da execução impede que esses casos vão adiante.

**switch** **(**n\_Registros**)**

**{**

**case** 100**:**

**return** 5**;**

**case** 1000**:**

**return** 50**;**

**case** 10000**:**

**return** 100**;**

**case** 100000**:**

**return** 200**;**

**case** 1000000**:**

**return** 500**;**

**}**

1. Definição da tabela e preenchimento

O código de exemplo fornecido lê cada registro e utiliza uma variável auxiliar para realizar um somatório e selecionar as chaves a serem adicionadas ao índice da tabela. Alteramos essa função de forma que apenas o inteiro referente ao índice daquela página seja lido no arquivo.

int aux**;**

**for** **(**int i **=** 0**;** fread**(&**aux**,** **sizeof(**int**),** 1**,** arquivo\_Binario**);** i**++)**

**{**

tabela\_Indice**[**i**]** **=** aux**;**

fseek**(**arquivo\_Binario**,** **(**itens\_Pagina **\*** **sizeof(**Registro**))** **-** **sizeof(**int**),** SEEK\_CUR**);**

**}**

rewind**(**arquivo\_Binario**);**

Após a leitura do inteiro, o ponteiro, na posição seguinte, é deslocado na quantidade de 1 página – 1 inteiro, posicionando-o sobre a primeira chave da página seguinte. Ao final do processo, o ponteiro é reiniciado na posição inicial do arquivo.

1. Busca pelo índice

Iniciamos tratando casos onde o registro obviamente não se encontra no arquivo. Em um arquivo crescente, por exemplo, se o registro for menor que a primeira chave, então a busca pode ser interrompida imediatamente. Como o enunciado também inclui a possibilidade de arquivos decrescentes, utilizamos a mesma lógica para excluir esse tipo de caso.

**switch** **(**n\_Situacao**)**

**{**

**case** 1**:**

**if** **(**tabela\_Indice**[**0**]** **>** n\_Chave**)**

**return** **-**1**;**

**for** **(**aux **=** 0**;** aux **<** tam\_Tabela**;** aux**++)**

**{**

**if** **(**tabela\_Indice**[**aux**]** **==** n\_Chave**)**

**return** aux**;**

**if** **(**tabela\_Indice**[**aux**]** **>** n\_Chave**)**

**return** aux **-** 1**;**

**}**

**return** aux **-** 1**;**

**break;**

**case** 2**:**

**if** **(**tabela\_Indice**[**0**]** **<** n\_Chave**)**

**return** **-**1**;**

**for** **(**aux **=** 0**;** aux **<** tam\_Tabela**;** aux**++)**

**{**

**if** **(**tabela\_Indice**[**aux**]** **==** n\_Chave**)**

**return** aux**;**

**if** **(**tabela\_Indice**[**aux**]** **<** n\_Chave**)**

**return** aux **-** 1**;**

**}**

**return** aux **-** 1**;**

**break;**

**}**

Em seguida, percorremos a tabela sequencialmente comparando os índices com a chave pesquisada. No caso de um arquivo decrescente, se a chave for maior do que o índice atualmente comparado, então ela só pode estar localizada na página anterior. Se a iteração for concluída, retornamos o índice da última página independentemente, pois se a chave existir no arquivo, então ela só pode pertencer àquela página.

1. Leitura da página e busca em memória principal

Uma vez encontrado o índice da página, calculamos o deslocamento, posicionamos o ponteiro do arquivo de acordo e realizamos a leitura dos dados. Em memória principal, outra pesquisa sequencial é realizada e a iteração é interrompida se a chave pesquisada for encontrada. Note que não há preocupação quanto à possibilidade de página incompleta, pois a dinamização do tamanho da página impede que isso aconteça para as possibilidades de entradas enunciadas no trabalho.

Registro aux**[**itens\_Pagina**];**

long int desloc\_Ponteiro **=** itens\_Pagina **\*** indice\_Pagina **\*** **sizeof(**Registro**);**

fseek**(**arquivo\_Binario**,** desloc\_Ponteiro**,** SEEK\_SET**);**

fread**(**aux**,** **sizeof(**Registro**),** itens\_Pagina**,** arquivo\_Binario**);**

**for** **(**int i **=** 0**;** i **<** itens\_Pagina**;** i**++)**

**{**

**if** **(**aux**[**i**].**chave **==** n\_Chave**)**

**{**

printf**(**"Registro encontrado!\n"**);**

imprimir\_Registro**(**aux**[**i**]);**

**return** 1**;**

**}**

**}**

1. Análise experimental

Para a análise experimental do método, utilizamos a função clock da biblioteca time.h para realizar o cálculo de tempo de execução dos métodos de pesquisa e variáveis inteiras para somar as comparações e leituras. Tanto as etapas de criação dos índices quanto a própria pesquisa foram contabilizadas, e as chaves pesquisadas foram determinadas por ‘1 + rand () % (nro\_registros – 1)’. Observamos que o número de comparações é variável, haja vista que depende do quão distante o índice da página e a própria chave estão em relação ao início dos vetores que as armazenam. Já o número de transferências é constante para o mesmo número de registros, pois a obtenção da tabela de índices efetua (nro\_registros / tam\_tabela) transferências de inteiros (chaves) e a pesquisa sequencial requer a transferência de uma página da grandeza de (itens\_pagina).