Universidade de São Paulo
Instituto de Ciências Matemáticas e de Computação

Trabalho 01 - Árvore B

SCC05003 - Algoritmos e Estrutura de Dados II
Professora: Elaine Parros Machado de Sousa
Estagiário PAE: Evandro Ortigossa

Patrick Oliveira Feitosa 10276682

São Carlos, São Paulo 08/06/2020

1. Introdução

Esse trabalho tem como principal objetivo a implementação de uma Árvore B como estrutura de organização de indices de um arquivo de dados. O tema dessa aplicação é um sistema de cadastro de alunos na qual o professor ou usuário pode fazer a inserção de alunos e sua informações, fazer busca de algum aluno específico ou visualizar todos os alunos cadastrados.

As Arvores B são um típo de árvore N-ária na qual em cada nó é possivel que haja mas de um elemento. As suas inserções são sempre feitas nas folhas permitindo que ela esteja balaceada, possuindo um crescimento no sentido ***bottom-up***. Cada nó dessa árvore pode ter até (ORDEM - 1) elementos, sendo a ORDEM igual a quantidade de nós filhos que esse nó pode ter.

2. Descrição das Implementações

O trabalho pode ser dividido em duas estruturas principais: Arquivo de Dados e Aquivo de Índices, os quais são explicitados abaixo.

a. Arquivo de Dados

Nesse arquivo estarão todos os alunos cadastrados em forma de registro. Nos registros são salvo informações de numero USP, nome, sobrenome, curso e nota do aluno. Essa estruturas são de campo e tamanho fixos e são organizados, utilizando *struct*, da seguinte forma:

```
typedef struct{

    int n_usp;
    char nome[10];
    char sobrenome[20];
    char curso[20];
    float nota;

}tipoAluno;
```

Figura 1: Estutura dos registros do aluno

O arquivo possui extensão de extensão de aquivo de texto possui o nome de "***dados.txt***". Cada registro é ordenado por ordem de inserção, sempre inserindo no final do arquivo.

b. Arquivo de Indices

Já esse aquivo é estururado em uma Árvore B, em vez de registros, chamamos de página ou nós. Cada página possui um contador, responsável por controlar a quantidade de indices atuais, um vetor de indices e um vetor com os RRNs das paginas filhas.

```
typedef struct {
    int contador;
    INDICE indices[ORDEM-1];
    long int RRN_filhos[ORDEM];
}PAGINA;
```

Figura 2: Estrutura da pagina

```
typedef struct {
    int chave;
    long int RRN_arq;
}
INDICE;
```

Figura 3: Estrutura dos indices

Para termos um grau de eficiencia consideravel, foi pedido que cada página tivesse o tamanho de um bloco de disco, que na maioria dos casos é em torno de 4Kb. Fazendo os cálculos obtivemos para ORDEM = 166 um tamanho de 3976 bytes. Escolhendo essa ORDEM, foi possivel ter 165 indices por pagina.

O arquivo de extensão binária, possui o nome de "***index.dat***", como foi explicitado no documento de requisistos. A primeira pagina desse arquivo conterá a cabeça do arquivo, nela terão informações de quantidade de indices e o RRN da raiz da árvore. Além disso, foi criado uma estrutura auxiliar chamada AUX, que será utilizada na inserção de um índice, contendo informações de indice e RRN de pagina.

```
typedef struct{
    INDICE indice_aux;
    long int RRN_n_filho;
}
AUX;
```

Figura 4: Estrutura das *struct* auxiliar

3. Funções

Aqui iremos explicar detalhamente o objetivo de cada função. A ordem seguida será a mesma que está no código fonte, para facilitar a visualização.

- ***void printarPagina(****PAGINA *pagina_aux****)***

  Parâmetros:

  pagina_aux: ponteiro para a pagina lida na memoria

  Função:

  Essa função tem o intuito de printar a pagina que está lida na memória apontada pelo ponteiro pagina_aux.

- ***void resetarPagina(PAGINA *pagina_aux)***

Parâmetros:

pagina_aux: ponteiro para a pagina lida na memoria

Função:

Essa função tem o intuito de zerar o contador na pagina_aux e setar -1 no vetor de RRNs.

- ***void gravarPagina(FILE *fi, PAGINA *pagina_aux, long int RRN)***

Parâmetros:

fi: ponteiro para o arquivo de índices
pagina_aux: ponteiro para a pagina lida na memoria
RRN: RNN da pagina no arquivo de indices

Função:

Essa função tem o intuito de gravar a pagina_aux na posicao RRN do arquivo de indices

- ***int carregarPagina(FILE *fi, PAGINA *pagina_aux, long int RRN)***

Parâmetros:

fi: ponteiro para o arquivo de índices
pagina_aux: ponteiro para a pagina lida na memoria
RRN: RNN da pagina no arquivo de indices

Função:

Essa função tem o intuito de ler para a posição da memória apontada por pagina_aux a pagina com esse RRN

- ***long int *percorrerArvore(FILE *fi, PAGINA *pagina_aux, long int RRN_Raiz, int chave, int flag)***

Parâmetros:

fi: ponteiro para o arquivo de índices
pagina_aux: ponteiro para a pagina lida na memoria
RRN_Raiz: RNN da raiz da arvore
chave: chave do indice
flag: indica se ele ira percorrer a arvore para inserção ou busca

Função:

Se flag == BUSCA, a função retorna o RRN da posicao do registro referente a essa chave ou -1 se a chave não foi encontrada. Se a flag == INSERCAO, a função retorna um um vetor com os RRNs das paginas percorridas e a posição de inserção na ultima página.*(Ex.: retorno = [4, 3, 0, 1], o ultimo número do retorno (1) é a posição de inserção do indice na pagina RRN = 0, o 3 é referente a pagina anterior(RRN_raiz = 3) e 4 é o tamanho do vetor de retorno)*

- ***int iniciarArvore(FILE *fi)***

Parâmetros:

fi: ponteiro para o arquivo de índices

Função:

Cria uma cabeça para o arquivo de indices, setando os valores de de qtd de indices e a posicao da raiz iguais a zero. Retorna zero se for concluida, c.c. um.

- ***long int* informacoesCabeca(FILE *fi, PAGINA *pagina_aux)***

Parâmetros:

fi: ponteiro para o arquivo de índices
pagina_aux: ponteiro para a pagina na memoria

Função:

Retorna um vetor com a qtd de indices inseridos e a posicão da raiz no aquivo de indices presentes na cabeca

- ***void inserirArvore_aux(FILE *fi, PAGINA *pagina_aux, long int* caminho, long int *info, int pos, AUX *auxiliar)***

Parâmetros:

fi: ponteiro para o arquivo de índices
pagina_aux: ponteiro para a pagina na memoria
caminho: ponteiro para o caminho percorrido até a posicao de inserção
info: ponteiro contendo as informações da cabeça
pos: posição do vetor caminho
auxiliar: ponteiro para estrutura auxiliar

Função:

Essa é uma função recursiva e tem o intuito de inserir o indice presente em **auxiliar** na pagina na posicão **caminho[pos]** na pagina com RRN **caminho[pos-1]**. Se ouver quebra de nó o indice e os RRN são repartidos entre pagina atual e uma nova que será criada, após isso, **auxiliar** recebe o indice mediano da pagina junto com o RRN da nova pagina alocada e uma nova chamada da função e tentar inserir na página anterior. Se houver quebra de nó na raiz é criado outra pagina para receber esse esse indice mediano.

- ***int inserirArvore(FILE *fi, PAGINA* pagina_aux, long int* info, long int RRN_arq, int chave)***

Parâmetros:

fi: ponteiro para o arquivo de índices
pagina_aux: ponteiro para a pagina na memoria
info: ponteiro contendo as informações da cabeça
RRN_arq: RRN do registro referente a chave
chave: chave que será adicionada

Função:

Adiciona uma chave junto com RRN do registro na arvore

- ***void buscarAluno(FILE *fp, FILE *fi, PAGINA *pagina_aux, long int *info)***

Parâmetros:

fp: ponteiro para o arquivo de dados
fi: ponteiro para o arquivo de índices
pagina_aux: ponteiro para a pagina na memoria
info: ponteiro contendo as informações da cabeça

Função:

Printa na tela a informações do aluno buscado pelo numero usp

- **void visualizarArvore(FILE *fi, PAGINA *pagina_aux)**

Parâmetros:

fi: ponteiro para o arquivo de índices
pagina_aux: ponteiro para a pagina na memoria

Função:

Printa o estado atual da arvore

- ***void atualizarCabeca(FILE *fi, PAGINA *pagina_aux, long int *info)***

Parâmetros:

fi: ponteiro para o arquivo de índices

pagina_aux: ponteiro para a pagina na memoria
info: informações atualizadas da árvore

Função:
Reescreve na cabeça os status atual RRN da raiz e a quantidade atual de índices

- ***void inserirAlunos(FILE *fp, FILE *fi, PAGINA* pagina_aux, long int* info, tipoAluno aluno)***

Parâmetros:
fp: ponteiro para o arquivo de dados
fi: ponteiro para o arquivo de índices
pagina_aux: ponteiro para a pagina na memoria
info: informações da árvore
aluno: informações do aluno

Função:
Insere os dados do aluno no arquivo de dados e no arquivo de índices

- ***void cadastrarAluno(FILE *fp, FILE *fi, PAGINA* pagina_aux, long int* info)***

Parâmetros:
fp: ponteiro para o arquivo de dados
fi: ponteiro para o arquivo de índices
pagina_aux: ponteiro para a pagina na memoria
info: informações da árvore

Função:
Lê as informações do aluno que será cadastrado

- ***void leituraAlunos(FILE *fp)***

Parâmetros:
fp: ponteiro para o arquivo de dados

Função:
Percorre o arquivo de dados printado as informações dos alunos cadastrados.

- **void criarArquivos(FILE *fp, FILE *fi)**

Parâmetros:
fp: ponteiro para o arquivo de dados

fi: ponteiro para o arquivo de índices

Função:

Cria os arquivos de dados e de indíces junto com a cabeça.

- ***void gerarDados(FILE *fp, FILE *fi, PAGINA *pagina_aux, long int *info, int qtd)***

Parâmetros:

fp: ponteiro para o arquivo de dados
fi: ponteiro para o arquivo de índices
pagina_aux: ponteiro para a pagina na memoria
info: informações da árvore
qtd: quantidade de estudantes que serão gerados

Função:

Gera uma quantidade de alunos aleatoriamente.

- ***void menu()***

Função:

Printa o menu.

4. Método de Compilação e Execução

Para compilar e executar o algoritmos basta execultar os seguintes

```
gcc main.c -o main -ansi /*  Compila  */
./main                   /*  Executa  */
```

Ao executar o programa, é gerado um quantidade inicial de alunos aleatório determinada na main do código.

5. Conclusão

Foi possível implementar corretamente a árvore como estrutura de índices. Além disso, percebe-se que os nós tendem a ter um grau de preenchimento de 50%, uma vez que no momento da quebra é elencado o elemento mediano para subir para o nó acima. Esse fato ocorreu claramente nao inserir os dados aleatóriamente, com chaves em ordem crescente