UNIVERSIDADE FEDERAL DE LAVRAS

TRABALHO

Algoritmo e Estrutura de Dados III - GCC109

Trabalho III

Alunos: Augusto Soares Pereira - 20120320
Álvaro dos Reis Cozadi - 201211012
Ueslei Marcelino da Guia - 201120619

Lavras - MG

2014

1. Introdução

Na Ciência da Computação, uma tabela dispersão (também conhecida por tabela de espalhamento, tabelas esparsas ou ainda tabela hash, do inglês hash) é uma estrutura de dados especial, que associa chaves de pesquisa a valores. A utilização de tabelas hash em estrutura de dados é algo importante e com aplicações em diversas áreas (utilizadas para implementar vetores associativos, conjuntos e caches, indexação de grandes volumes de informação, etc), visto que fornecem um acesso muito mais rápido aos elementos de um vetor, por exemplo, que qualquer outro método (listas encadeadas, árvores binárias, etc). Seu objetivo é, a partir de uma chave simples, fazer uma busca rápida e obter o valor desejado.

A implementação típica busca uma função de dispersão (função hash) que seja de complexidade O(1), não importando o número de registros na tabela (desconsiderando colisões). O ganho com relação a outras estruturas associativas (como um vetor simples) passa a ser maior conforme a quantidade de dados aumenta. A função hash é a responsável por gerar um índice a partir de determinada chave. Caso a função seja mal escolhida, toda a tabela terá um mau desempenho. O ideal para a função hash é que sejam sempre fornecidos índices únicos para as chaves de entrada. A função perfeita (hashing perfeito) seria a que, para quaisquer entradas A e B, sendo A diferente de B, fornecesse saídas diferentes.

2. Organização do código

O código está dividido em 4 arquivos. No arquivo `hash.c` estão implementados as função de manipulção e inicialização do hashing que irár em cima dos dados. Já no arquivo `hash.h` estão definidos macros que serviram de controle no programa, as estruturas de cabeçalho, de índices e de aluno e também as funçôes de todo o programa. O arquivo `main.c` corresponde ao arquivo principal com a função main e

interface implementadas, nos fornecendo a possibilidade inserir um aluno novo, realizar uma busca tanto por ID como por MAT e também atualizar os seus dados.

3. Funcionalidade Detalhadas

Escolhemos descrever cada arquivo detalhadamente para facilitar o seu entendimento.

3.1 hash.h

No arquivo hash.h estão implementadas algumas contantes de controle que iram facilitar o controle e também à legibilidade de todo o código.

Também estão presentes 3 estruturas com funções distintas. São elas:
Estrutura responsável por armazenar informações sobre o número de posições ativas e a quantidade total de dados.

```
typedef struct CabecalhoDados
{
    int Total;
    int Ativos;
} CabecalhoDados;
```

Estrutura de cada bloco onde serão usados dois arquivos hashing, um organizado pelo número de matrícula e outro organizado pelo número da identidade.

```
typedef struct Bloco
{
    char Chaves[TAM_BLOCO][15];
    int RRNs[TAM_BLOCO];
} Bloco;
```

Estrutura referente a Aluno, que deverar ter um nome, uma identidade, um CPF, uma matrícula e o RSG (CRA).

```
typedef struct Aluno
{
    char Nome[50];
    char Identidade[TAM_CHAVE + 1];
    char CPF[15];
    char Matricula[TAM_CHAVE + 1];
    double RSG;
} Aluno;
```

No fim do arquivo, estão declaradas todas as funções que serão utilizadas para manipular todos os dados ao longo da execução do programa.

3.2 hash.c

No arquivo foi implementada a função hashing que retorna a posição onde o aluno deve ser armazenado.

- Uma função de busca bloco que precisa de três parâmentros, a chave do bloco, o RRN do bloco e o RRN do primeiro bloco encontrado com uma chave removida a partir da primeira chamada. Essa função tem três tipos de retorno, NAO_ENCONTROU_CHAVE, ENCONTROU_CHAVE e HASH_CHEIO.
- Uma função que busca um bloco expecifico no arquivo, que necessita do número RRN e retorna um ponteiro para um bloco específico.
- Uma função que recebe um bloco e o seu RRN, onde será realizada a escrita neste bloco.
- Uma função que irár buscar uma determinada chave passada como parâmetro. A função apresenta três tipos de retorno, HASH_CHEIO, NAO_ENCONTROU_CHAVE, a chave do bloco solicitada e por ultimo -1 caso aconteça algum erro.

- Uma função que realiza a inserção de alguma chave passada como parâmentro, com quatro tipos de retorno HASH_CHEIO, ENCONTROU_CHAVE, NAO_ENCONTROU_CHAVE e SUCESSO caso a inserção tenha ocorrido.
- Uma função denominada Drive que faz o carregamento de todos os arquivos e índices.
- Uma função que realiza a remoção de uma chave desejada, tendo como retorno quatro tipos de valores, HASH_CHEIO, NAO_ENCONTROU_CHAVE, RRN e -1.
- Uma função que irá imprimir todos os alunos cadastrados com o seu nome, seguido pela identidade, CPF, número de matrícula e RSG.
- Uma função que irár ler do disco os dados dos alunos para poderem ser escritos na tela para o usuário.
- Por fim, duas funções que realizam a impressão dos blocos como solicitado na descrição do trabalho.

3.3 - main.c

Neste arquivo principal estão implementadas a função main e a interface. A sua função é fornecer ao usuário a capacidade de inserir um novo aluno, remover um aluno tanto por identidade como por matrícula. Capacidade de realizar uma busca por identidade e por matrícula e também a possibilidade de atualizar os dados ja inseridos no arquivo.

3.3.1 - Inserir um novo Aluno

Implementação do código responsável na inserção de um novo aluno. O código recebe os dados válidos de aluno para serem gravados na árvore e por fim, gravados no disco. Linhas 46 a 84 no arquivo **main.c**.

3.3.2 - Remover um Aluno

A remoção pode ser feita de duas formas. Pode ser realizada pelo número de identificação do aluno inserido ou pelo número de matrícula do aluno.
Primeiro o algoritmo faz a identificação de qual opção de remoção foi digitada e depois realiza a busca pelos dados do aluno a ser removido. Linhas 91 a 115 a remoção é por identidade. E linhas 116 a 143 a remoção é feita por matrícula. Ambas informações se encontram no arquivo **main.c**.

3.3.3 - Busca por Aluno

A busca pode ser feita de duas formas. Pode ser realizada pelo número de identificação do aluno inserido ou pelo número de matrícula do aluno.
O algoritmo faz a identificação de qual opção de busca foi digitada e depois realiza a busca pelos dados do aluno. Linhas 151 a 169 a busca é por identidade. E linhas 170 a 188 a busca é feita por matrícula. Ambas informações se encontram no arquivo **main.c**.

3.3.4 - Atualizar os dados

Na atualização dos dados do alunos já cadastrado, o algoritmo precisa de novos valores validos concretizar a atualização. A atualização pode ser feita de duas formas. Pode ser realizada pelo número de identificação do aluno inserido ou pelo número de matrícula do aluno. Linhas 197 a 232 a atualização é por identidade. E linhas 233 a 268 a atualização é feita por matrícula. Ambas informações se encontram no arquivo **main.c**.

3.4 - Makefile

O Makefile foi criar para automatizar a compliação e execução do programa. Ele aprensenta dois comando apenas, `make` para compilar e `make clear` para apagar todos os arquivos .dat.

4. Exemplos

Exemplo 1: Inserção de alguns valores.

```
INS Augusto 123 123 123 7.8
SUCESSO
INS Pedro 321 321 321 4.5
SUCESSO
INS Thais 444 444 444 8.9
SUCESSO
INS Luiza 876 908 543 8.3
SUCESSO
```

Exemplo 2: Exemplo de saída realizada pelo comando EXI ID

```
EXI ID
[-1|-1|-1]
[321,1|-1|-1]
[-1|-1|-1]
[123,0|-1|-1]
[444,2|-1|-1]
[-1|-1|-1]
[876,3|-1|-1]
[-1|-1|-1]
[-1|-1|-1]
[-1|-1|-1]
```

Exemplo 3: Exemplo de saída realizada pelo comando EXI MAT

```
EXI MAT
[-1|-1|-1]
[321,1|-1|-1]
[-1|-1|-1]
[123,0|-1|-1]
[444,2|-1|-1]
[-1|-1|-1]
[-1|-1|-1]
[-1|-1|-1]
[-1|-1|-1]
```

```
[-1|-1|-1]
```

Exemplo 4: Exemplo de saída do comando BUS ID

```
BUS ID 123
<Augusto> <123> <123> <123> <7.80>
```

Exemplo 5: Exemplo de saída do comando BUS MAT

```
BUS MAT 444
<Thais> <444> <444> <444> <8.90>
```

5. Mecanismos

A codificação feitas para atender os requisitos solicitados pelo trabalho, foram feitas na IDE Geany 0.21 e editor de texto Atom version 0.158.0, ambas utilizando o S.O Ubuntu 14.04.

O código deve ser executado pelo terminal de um Sistema Operacional GNU/Linux utilizando o comando make.

O projeto apresenta um arquivo denominado `Makefile` com dois comandos, o `make all` para compilar todos os arquivos e juntar todos os arquivos objeto. E o segundo e último comando, `make clear`, que realiza a exclusão de todos os arquivos do tipo `.dat` presentes na pasta do projeto.

A execução do trabalho deve ser realizada utilizando o compilador GCC da seguinte forma:

1. Passo 1 - `make`
2. Passo 2 - `./Sysalunos <nome_arquivo>.dat`
3. Passo 3 - `Entrada dos dados`

Para excluir todos os arquivos `.dat` da pasta onde está localizado o projeto, basta uitlizar comando `make clear`.