Fabio Fogarin Destro, Paulo A. de Oliveira Carneiro, Renata Vinhaga dos Anjos, Vítor H. Gratiere Torres

Primeiro Trabalho Prático de Organização de Arquivos

Professora Dra. Cristina Dutra de Aguiar Ciferri

São Carlos, SP

Maio/2017

Sumário

1 Introdução

Este trabalho consiste na implementação de 9 funções que envolvem manipulação de dados. Dentre essas são: inserção, remoção e atualização de dados baseados na abordagem dinâmica de reaproveitamento de espaços de registros logicamente removidos.

O programa foi feito utilizando a linguagem `C`, sendo compilado e testado no sistema operacional Linux (`Ubuntu 16.04 lts`).

Pode-se garantir que o arquivo de dados ainda não exista antes da primeira execução: `make clean`.

Para compilar: `make all`.

Para executar o programa: `./arquivo [FUNCAO] [...ARGUMENTOS...]`.

2 Macros

Utilizamos algumas substituições de sintaxe para facilitar o entendimento e a generalização do código. Essas macros foram usadas no cálculo do *Byte Offset*:

- `TAM_CAMPO` - tamanho dos campos de tamanho fixo (10 bytes);
- `TAM_TEG` - tamanho fixo para os registros do trabalho (112 bytes) como especificado;
- `TAM_CABECALHO` - tamanho do cabeçalho do arquivo binário (5 bytes = 1 byte para status do arquivo + 4 bytes para o topo da pilha de remoção lógica).

3 Estrutura de Dados

Usamos duas `structs` para auxílio e armazenamento temporário de dados para manipulação:

- `Cabeçalho`: contém o status do arquivo `saida.bin` (`0` para inconsistente e `1` para consistente) e o topo da pilha de registros logicamente removidos (`-1` quando não há registros removidos);
- `Escola`: contém o código da escola e cinco ponteiros para as datas de Início e Final do ano letivo escolar (de tamanho fixo), para os nomes da Escola, do Município e do Endereço, precedidos de seus indicadores de tamanho.

4 Função 1

`Função [1]` - Lê vários registros a partir de um arquivo de entrada CSV e grava esses registros em um arquivo de dados de saída. Única função que se utiliza do arquivo `.csv`, todas as posteriores manipulam o arquivo de saída `saida.bin` gerado por esta primeira.

A partir do nome do arquivo de entrada e o nome do arquivo de saída, é executada a leitura do arquivo de formato `.csv` fornecido com argumento de entrada do programa e a gravação dos registros no arquivo de dados gerado como saída.

O algoritmo implementado consiste nos seguintes passos:

1. Abre o arquivo de entrada em modo de leitura e cria ou estende o arquivo de saída em modo de leitura e escrita binária;
2. Cria o registro de cabeçalho, no início do arquivo de saída;
3. Lê linha a linha (registro a registro) do arquivo de entrada, armazenando os campos em uma estrutura auxiliar (`struct Escola`) que será utilizada para escrever no arquivo de saída;
4. Cada campo é separado ao percorrer a linha e encontrar o `char ';'` ou `char '\n'` (fim de linha) que delimita cada campo, no caso dos campos de tamanho fixo, seu valor é salvo na `struct`, já no caso de campos de tamanho variado, é necessário também que o seu tamanho seja contabilizado e salvo em um indicador de tamanho correspondente;
5. Com a `struct Escola` preenchida contendo os valores dos campos e dos indicadores de tamanho quando necessário, a escrita no arquivo de saída é efetuada;
6. Como os registros são de tamanho fixo, é necessário completar o tamanho do registro caso seja necessário, para que ele tenha constantemente 112 bytes, então após a escrita de cada campo e indicadores de tamanho para campos de tamanho variado, são adicionados bytes nulos até o tamanho do registro alcançar seu tamanho predeterminado.
7. Atualiza o status do arquivo de saída para consistente e o fecha.

5 Função 2

`Função [2]` - Recupera dados de todos os registros, exibindo-os de forma organizada.

A partir do nome do arquivo de dados, todos os registros são recuperados e exibidos de forma organizada, imprimindo cada registro em uma linha, separando os campos por espaços e imprimindo também, antes dos campos de tamanho variado, seu tamanho em bytes.

O algoritmo implementado consiste nos seguintes passos:

1. Abre o arquivo de dados em modo de leitura binária `saida.bin`, verifica se está consistente e atualiza seu status para inconsistente.
2. Pula os 5 bytes iniciais, referentes ao registro de cabeçalho;
3. Para cada registro (112 bytes), é verificado se o registro foi ou não logicamente removido, ou seja, os 4 primeiros bytes são verificados se são iguais a `-1`;
4. Para cada registro não removido, cada campo é lido e salvo na `struct Escola auxiliar`, em que campos de tamanho fixo são lidos utilizando o `TAM_CAMPO` e os de tamanho variável utilizando o indicador de tamanho;
5. Após ler todos os campos, é preciso pular os possíveis bytes nulos que completam o registro para que ele seja de tamanho fixo, e dessa forma para cada registro 112 bytes sempre são percorridos no arquivo de dados;
6. Imprime o conteúdo da `struct auxiliar` que contém um registro, imprimindo cada campo separado por um espaço e para campos de tamanho variável o seu indicador de tamanho em bytes;
7. Faz isso até o fim do arquivo de dados;
8. Atualiza o status do arquivo de saída para consistente e o fecha.

6 Função 3

`Função [3]` - Recupera todos os dados que satisfazem um critério de busca determinado pelo usuário e os imprime de forma organizada.

O algoritmo implementado consiste nos seguintes passos:

1. Abre o arquivo de dados em modo de leitura binária `saida.bin`, verifica se está consistente e atualiza seu status para inconsistente.
2. Abrir o arquivos de dados no modo de leitura binária;
3. Pular os 5 bytes iniciais, referentes ao registro de cabeçalho;

4. Para cada registro (112 bytes), é verificado se o registro foi ou não logicamente removido, ou seja, os 4 primeiros bytes são verificado se são iguais a -1;

5. Para cada registro não removido, seus dados são salvos na struct auxiliar Escola e é verificado se o registro satisfaz o critério de busca do usuário;

6. Cada campo que satisfaz o critério de busca é exibido na tela, com os campos separados por espaço.

7. Atualiza o status do arquivo para consistente e o fecha.

7 Função 4

`Função [4]` - Recupera um registro do arquivo de dados por seu RRN fornecido pelo usuário.

1. Abre o arquivo de dados em modo de leitura binária `saida.bin`, verifica se está consistente e atualiza seu status para inconsistente.

2. Com o RRN fornecido, calcula o byte offset pela equação:

$$\texttt{offset = (RRN * TAM\_REG) + TAM\_CABECALHO} \tag{1}$$

3. Dado o deslocamento no arquivo, verifica-se se o registro pedido já está removido. Caso não esteja, recupera o registro passando cada campo para a estrutura Escola e o imprime;

4. Atualiza o status do arquivo para consistente e o fecha.

8 Função 5

`Função [5]` - Remove logicamente um registro do arquivo de dados pelo RRN.

1. Abre o arquivo de dados em modo de leitura binária `saida.bin`, verifica se está consistente e atualiza seu status para inconsistente.

2. Recupera o topo da pilha de remoção lógica;

3. Com o RRN fornecido, calcula o byte offset pela Equação 1 na página 6;

4. Dado o deslocamento no arquivo, verifica-se se o registro já está removido. Caso não esteja, sobrescreve os primeiros 4 bytes do registro com o indicador de remoção `-1` e os próximos 4 bytes com o topo da pilha;

5. Agora o novo topo da pilha será o RRN do registro que foi removido (caso tenha sido);

6. Atualiza o status do arquivo para consistente e o fecha.

9 Função 6

`Função [6]` - Insere um novo registro de acordo com a dinâmica de reaproveitamento de espaços de registros logicamente removidos.

1. Abre o arquivo de dados em modo de leitura e escrita binária `saida.bin`, verifica se está consistente e atualiza seu status para inconsistente.

2. Recupera o topo da pilha de remoção lógica;

3. Se a pilha estiver vazia o novo registro será inserido no final do arquivo de dados `saida.bin`. Se não, é feito o cálculo do byte offset para inserção pela Equação 1 na página 6, de acordo com o local do último registro logicamente removido;

4. Recupera-se, do registro removido, o novo topo da pilha;

5. Os dados do novo registro, passados por argumento, são armazenados temporariamente na estrutura de dados Escola e posteriormente escritos no arquivo de saída `saida.bin`;

6. O novo topo da pilha é inserido no `Cabeçalho` do arquivo `saida.bin`;

7. Atualiza o status do arquivo para consistente e o fecha.

10 Função 7

`Função [7]` - Atualiza um registro, caso existente, por seu RRN.

1. Abre o arquivo de dados em modo de leitura e escrita binária `saida.bin`, verifica se está consistente e atualiza seu status para inconsistente.

2. Com o RRN fornecido, calcula o byte offset pela Equação 1 na página 6;

3. Dado o deslocamento do arquivo, verifica a existência do registro pelo indicador de remoção (`-1`). Caso não esteja removido os dados de atualização do registro, passados por argumento, são armazenados temporariamente na estrutura de dados `Escola` e posteriormente escritos no arquivo de saída `saida.bin`;

4. Atualiza o status do arquivo para consistente e o fecha.

11 Função 8

`Função [8]` - Realiza a desfragmentação do arquivo de dados `saida.bin`.

1. Abre o arquivo de dados em modo de leitura binária `saida.bin`, verifica se está consistente e atualiza seu status para inconsistente.
2. Muda o nome do arquivo `saida.bin` para `antigo.bin`, cria um novo arquivo chamado `saida.bin` (mesmo nome do arquivo original). Cria um novo `Cabeçalho`, definindo seu status para inconsistente;
3. Percorre o arquivo original, registro a registro, até chegar ao final, processando apenas aqueles que não estejam removidos. Caso um registro não esteja removido, recupera-o passando seus campos para a estrutura de dados `Escola`, e posteriormente escrevendo no novo arquivo `saida.bin`;
4. Remove o arquivo original (`antigo.bin`), pois temos um novo arquivo com o nome original (`saida.bin`) agora desfragmentado;
5. Atualiza o status do novo arquivo para consistente e o fecha.

12 Função 9

`Função [9]` - Recupera os RRNs da pilha de registros logicamente removidos.

1. Abre o arquivo de dados em modo de leitura binária `saida.bin`, verifica se está consistente e atualiza seu status para inconsistente.
2. Recupera o topo da pilha de registros logicamente removidos no status do arquivo `saida.bin`;
3. Caso a pilha não esteja vazia (`topo != -1`), calcula o byte offset dos registros removidos pela Equação 1 na página 6, como exemplificado pela Figura 1 na página 9;
4. Recupera o próximo topo da pilha, localizado após o indicador de registro removido (`-1`), e executa o passo 3 novamente até que o topo seja vazio, sempre imprimindo topo;
5. Atualiza o status do novo arquivo para consistente e o fecha.

Figura 1 – Representação do algoritimo para recuperação da pilha