Universidade Federal de São Carlos
Campus Sorocaba

Estrutura de Dados 2

Indexação de Arquivos

Dupla:
Marcos Cavalcante Barboza
Yanick Oliveira Rossi

Professor:
Prof. Dr. Tiago Almeida

2º semestre de 2015

Sumário

1 Introdução

O objetivo desse trabalho é implementar um sistema que simula um bolão para apostas no jogo League of Legends. Entre as funcionalidades implementadas por esse sistema estão: Cadastro das partidas, Alteração das Partidas, Consulta (baseado em algum campo do registro) e Remoção de Registros.

Esse sistema de cadastro de partidas, tem como finalidade a prática de conceitos vistos em sala de aula no que tange a manipulação de arquivos e como indexá-los para poder recuperar/armazenar informação posteriormente.

2 Metodologia de Desenvolvimento

Para realizar a manipulação dos registros dentro do arquivo de dados e índices, nós procedemos da seguinte maneira:

Os registros foram armazenados em campos com tamanho fixo e variável (mesmo que em essência, todos os campos tem tamanho fixo), com o delimitador '@' para separar os campos dos registros e o caracter para preencher eventuais espaços não utilizados.

Para realizar a recuperação da informação (registros), utilizamos uma abordagem na qual o arquivo de dados é mantido em memória persistente e os arquivos de índice primário e secundários são carregados em memória principal (Memória RAM) para futura manipulação (buscas, por exemplo).

3 Processo de Desenvolvimento

Primeiramente, ao começar, o programa abre o arquivo de dados, e caso não exista, ele cria um, o arquivo de índices é carregado conforme o número de registros encontrados no arquivo de dados; Por exemplo, se não existe nenhum registro no arquivo de dados, não é criado um arquivo de índice nem índice primário, nem secundário, pois não temos nenhum registro (propriedade de não-criação).

Caso exista algum registro previamente cadastrado no arquivo de dados, o programa irá carregar os três arquivos de índice que têm (iprimary, winner e mvp) em memória principal. Para garantirmos a consistência dos dados em caso de eventuais falhas, uma flag (número 1 ou 0) indica se o arquivo do respectivo índice está atualizado (1) ou não (0). Se o arquivo de índice estiver inconsistente, eles são refeitos, partindo do arquivo de dados.

- O cadastro de registros é feito de maneira trivial, armanenando (em variáveis) o que é informado pelo usuário e posteriormente o cadastro é feito no arquivo.

Apesar de ser trivial, enfrentamos certos obstáculos quando redirecionavamos um arquivo de entrada para o nosso programa (./main ¡ cadastroTeste.in), pois o mesmo tinha algum tipo de formatação ou codificação diferente (muito por causa do sistema operacional ou do editor de textos), por isso, gastamos muito tempo tentando encontrar bugs nesse sentido. Finalmente, quando encontramos o bug, conseguimos usar o redirecionamento dos nossos arquivos de teste para futura comparação com os nossos arquivos de saída, tanto os providos pelo professor, quanto os nossos próprios.

- Para realizar a alteração de um registro, nós buscamos esse registro pelo código informado pelo usuário e conferimos na nossa lista de índice primário (que está em memória principal) para ver se tal registro existe, caso exista, nós retornamos o RRN (Record Relative Number) e procedemos com a busca no arquivo de dados; recuperamos o registro, alteramos o campo pedido pelo usuário (duração) e escrevemos no arquivo, mas agora com o campo (duração) alterado. A complexidade desejável dessa busca, seria $O(logn)$ comparações no indice primário (que está ordenado, uma busca binária deve ser usada usada) + o SEEK no arquivo de dados $O(1)$ em termos de acesso a memória seceundária. No entanto, nosso trabalho está usando uma busca sequencial $O(n)$ e o SEEK com $O(1)$, uma melhora interessante seria realizar a busca binária no arquivo de indices, poderíamos ter usado a busca binária da linguagem C, mas não tínhamos conhecimento dessa funcionalidade da biblioteca e preferimos, primeiro, deixar o programa funcionando e depois otimizá-lo. Todavia, temos plena noção da melhora substancial que tal busca teria em arquivos/lista de índice maiores.

- A remoçao é feita de maneira semelhante, porém, alteramos apenas dois caracteres no arquivo de dados e na lista de índice primário, buscamos o registro pelo código e setamos o RRN com -1. Custo: busca no arquivo de índice $O(n)$ + SEEK no arquivo de dados $O(1)$ acessos.

- As buscas são basicamente de dois tipos: chave primária (código) e chave secundária. A busca por chave primaria é feita diretamente na lista de índices primários, e depois um SEEK no arquivo de dados com o resultado da busca é feito. No caso da busca por chave secundária, a busca é feita no arquivo de chave secundária (Equipes vencedoras ou MVPs) com custo computacional de $O(n)$ + busca com o código retornado na lista de índice primário $O(n)$ + SEEK no arquivo de dados $O(1)$ acessos.

- A listagem é feita de maneira análoga a busca.

- A operação de liberar espaço, talvez, seja uma das mais custosas, pois percorremos o arquivo de dados buscando registro a registro e armazenamos esse registros encontrados em um outro arquivo de dados temporário, durante a liberação de espaço, também refazemos os indices primários e secundários, alterando o RRN no índice primário e como consequência nos dois indíces secundários também. O uso de dois arquivos, um temporário e outro real, aumenta e muito o custo computacional, porém não conseguimos encontrar uma maneira melhor de fazer isso nem uma função pertinente foi encontrada.

- Temos também funções auxiliares, como as usadas para ordenar as listas de índices primário e secundários, o algoritmo usada para tais ordenações é o QuickSort da biblioteca padrão do C. Esse algoritmo, quicksort, tem complexidade da ordem de $O(n * logn)$ comparações no caso médio e no pior caso tem algo da ordem $O(n^2)$ comparações.

3.1 Testes e Resultados

Realizamos vários testes no programa - inserção, remoção, consulta e alteração. Todos esses testes foram feitos num laptop MacOS X (UNIX) com gcc 4.9.2 rodando e também em um Ubuntu 15.04 com gcc 4.4.2, para isso criamos um conjunto de arquivos de teste próprio que está na pasta "arquivosDeTeste"e foi enviado em conjunto com os demais arquivos, não nos baseamos fielmente aos arquivos de teste fornecido, pois os mesmos apresentavam inconsistências. Para performar os teste, usamos redirecionamento dos mesmos via linha de comando ou rodamos um de nosso *Shell* scripts, que também seguem em anexo.

Criamos uma série de arquivos de testes próprios com a finalidade de testar a corretude de nosso programa, em todos os casos criados, nosso programa rodou e apresentou os resultados desejados e esperados. Ressaltamos que dependendo da sequência de operações, o programa pode não responder de maneira espera, mais uma vez, testamos o maior número de combinações possíveis e garantimos a corretude e integridade dos dados.

- **cadastro1OK.in** - Esse arquivo de teste cadastra 3 registros sem nenhum tipo de erro (tanto formatação dos dados).
- **cadastro2ER.in** - Esse arquivo de teste cadastra 2 registros com 7 erros de formatação de dados e .
  - 2 erros com o nome da equipe Azul (estouram o buffer).
  - 2 erros com o formato das datas
  - 1 erro com o formato da duração

  - 1 erro com o formato do placar da partida
  - 1 erro de reinserção

- **listagem1OK.in** - listagem dos três modos: Código, Equipe Vencedora e MVP.
- **alterar1OK.in** - Altera o registro (CCGE2310) para 10:59 e lista por codigo.
- **alterar2ER.in** - Altera o registro (CCGE2310), erra 4 vezes, para 10:10 e depois lista por codigo.
- **Cadastro2OK.in** - Cadastra 3 partidas com 4 erros; Lista por: Codigo e Nome da Equipe Vencedora; Busca por Nome da Equipe Vencedora (CHARMELEON) e busca por MVP
- **entradaGeral.in** - testa diversas funcionalidades do sistema.

4 Dificuldades Encontradas

A primeira dificuldade foi em relação a inserção de dados no arquivo, pois certas anomalias estavam acontecendo, tais como - quebras de linha e caracteres mal formatados, por isso, utilizamos por um bom tempo o modo copiar-colar ou apenas a inserção manual (digitando), finalmente achamos o bug e conseguimos fazer o redirecionamento de arquivos.

Certos bugs no código, tais como inserções do caractere por certas funções da biblioteca String do C, fizeram com que muito tempo fosse gasto no *debugging*.

Muita outros fatores externos foram barreiras para o desenvolvimento desse programa, podemos destacar: os trabalhos de outras matérias e provas de outras disciplinas que coincidiram de cair na mesma semana de entrega desse projeto.

5 Compilando e Rodando o Programa

Para executar o programa, você deve usar o comando *make*, que irá gerar um arquivo executável, que deve ser rodado da seguinte maneira: *./main ¡ arquivoTesteRedirecionado.in*.

Além disso, caso você queira rodar os arquivos de teste providos pela dupla, use: *chmod + testes.sh* esse comando irá permitir que o *Shell* script seja executado; por último, rode *./testes.sh*.

Dentro da pasta "arquivosDeTeste"será colocado os arquivos **.out** gerados dos testes.

Finalmente, para excluir os arquivos .o , de dados e a saída dos testes do grupo, use: *make clean*.