

Banco de dados em memória secundária

Clubes de futebol - AEDs III

Bruno R. Faria¹, Breno Lopes²

¹Ciências da Computação - Pontifícia Universidade Católica de Minas Gerais

Resumo. *Este documento descreve como foi possível a montagem de um sistema em terminal, simulando o acesso a um banco de dados. Capaz de adicionar times e fazer operações CRUDs no mesmo. O intuito do trabalho foi o entendimento de memória secundária, arquivos de dados estruturados e arquivo sequencial e indexado. Além de implementar uma lista invertida, uma busca binária para substituir as buscas sequenciais e uma ordenação interna no arquivo de índices, para sempre manter a ordenação. O trabalho foi realizado em linguagem de programação Java com controle de versões pela plataforma GitHub. O projeto é composto por três classes CRUD.java, Main.java e Clubes.java, os quais serão descritos no documento.*

1. Introdução

Para o início do desenvolvimento do projeto foi feita a seguinte pergunta: É necessário que se tenha a criação dos arquivos utilizados no programa ao iniciar, caso o arquivo não exista? A resposta para a pergunta foi simples, sim por conta de erros que poderiam ocorrer. Para iniciar o programa foi feito dentro dos construtores das classes utilizadas um código capaz de criar os arquivos necessário para todo o programa funcionar, o qual serviria de banco de dados para todos os dados inseridos pelo usuário, o da lista invertida serviria para armazenar a lista e o da ordenação externa e arquivo de índices para fazer a ordenação em memória secundária e para armazenar os dados do arquivo original, respectivamente. Com os arquivos criados, poderia dar início ao projeto com maior tranquilidade, tanto para os programadores quanto para o próprio usuário.

2. Desenvolvimento

Para iniciar o desenvolvimento, após a criação do construtor da classe CRUD no qual iria inicializar o arquivo e a inserção do ID = -1 no início do arquivo, deveria ser feita a classe Clube.java, na qual seria importante criar todos os atributos de um clube de futebol e as funções que teriam interações com o clube.

Os tipos de dados utilizados nas variáveis foram escolhidos de uma forma e não utilizar uma quantidade de Bits maior do que a necessária, um exemplo é o atributo ID onde foi colocado como byte, pois um campeonato de futebol como por exemplo, o Campeonato Brasileiro Série A, não possui mais que 20 times. Com isso, pode-se utilizar um tipo de dado que abrange uma quantidade menor de Bits.

Além do construtor da classe Clube, o arquivo conta com as funções "byte[] toByteArray()", a qual transforma os dados inseridos pelo usuário em um array de bytes a ser inserido no arquivo e retorna o mesmo, "void fromByteArray", recebe um array de bytes a ser lido pelo DataInputStream, "byte getId()" a qual retorna o id no clube, "void

```

public class Clube {
    protected byte id;
    protected String nome;
    protected String cnpj;
    protected String cidade;
    protected byte partidasJogadas;
    protected byte pontos;

    public Clube(byte i, String n, String cnpj, String c) {
        this.id = i;
        this.nome = n;
        this.cnpj = cnpj;
        this.cidade = c;
        this.partidasJogadas = 0;
        this.pontos = 0;
    }
}

```

Figure 1. Construtor e atributos dos clubes

increaseMatches()” a qual incrementa a quantidade de partidas jogada pelo clube e ”void updPoints(int x)” a qual recebe a quantidade de pontos a serem incrementados no clube, dependendo do resultado da partida (3 pontos para vitória e 1 ponto para derrota).

2.1. Interação com usuário

Inicialmente, o programa já cria a base de dados, com o nome ”clubes.db”, além de pegar o ultimo ID inserido no arquivo para que não seja necessário uma navegação geral do arquivo. O programa então, mostra para o usuário um menu, composto pelas seguintes opções ”1 - Criar time”, ”2 - Criar partida”, ”3 - Procurar time”, ”4 - Deletar time”, ”5 - Atualizar dados do time”, ”6 - Listar todos os times”, ”7 - Buscar na lista Invertida” e ”0 - Sair”.

Ao usuário selecionar a opção 1: Ele deve inserir todos os dados necessários para a criação de um clube (Nome, CNPJ, cidade), as partidas jogadas e os pontos não são inseridas pelo usuário mas são atualizadas ao usuário criar uma partida. Ao usuário inserir esses dados um objeto Clube é criado e a função ”create” disponível no arquivo CRUD.java é chamada, passando por parametros o objeto Clube criado e o id que foi incrementado ao criar o objeto.

Ao selecionar opção 2: O usuário deve inserir o nome de dois times e a quantidade de gols feitos na partida para cada um dos dois, ou seja, o resultado da partida. Caso os nomes dos times inseridos sejam válidos o programa chama a função ”matchGenerator”, disponível no arquivo CRUD.java, passando por parâmetro os nomes e os gols feitos por cada time.

Ao selecionar opção 3: O usuário insere um ID que deseja pesquisar e o programa chama a função ”readById”, disponível no arquivo CRUD.java, passando por parâmetro o id a ser pesquisado no arquivo ”clubes.db”.

Ao selecionar opção 4: O usuário insere um ID que deseja deletar e o programa chama a função ”delete”, disponível no arquivo CRUD.java, passando por parâmetro o id a ser deletado no arquivo ”clubes.db”, caso o clube não seja achado o programa retorna uma mensagem de erro para o usuário.

Ao selecionar opção 5: O usuário insere um ID que deseja atualizar, com isso

o programa irá chamar a função "readById", caso o ID seja válido o programa pedirá para que o usuário insira os novos dados que deseja para o clube, com isso um objeto Clube temporário e criado fazendo com que a função "update" seja chamada, passando por parâmetro o objeto Clube temporário a ser atualizado no arquivo "clubes.db", caso o não seja possível atualizar os dados do clube, uma mensagem de erro e retornada para o usuário, todas as funções chamadas estão disponíveis no CRUD.java.

Ao selecionar opção 6: O programa chama a função "readAll" disponível na classe CRUD.java e retorna todos os clubes disponíveis no arquivo "clubes.db" de forma sequencial.

Ao selecionar a opção 7: O programa deve dar o usuário a opção de buscar ids relacionados a uma busca por nome ou cidade no arquivo da lista invertida, mostrando assim todos ids relacionados aquela palavra.

Ao selecionar opção 0: O usuário passa para o programa o valor 0, o que significa que o laço de repetição While irá ser finalizado e o programa também.

2.2. Funções CRUD

2.2.1. Create

Ao receber os dados (objeto Clube e id atualizado que foi lido no arquivo) é dado início ao processo de criação do clube, o qual declara um vetor de bytes (ba), abre o arquivo para escrita, passa todas as informações que foram inseridas pelo usuário e armazenadas no objeto Clube para o vetor de bytes por meio do método da classe Clube, "toByteArray()", que, usando as classes "ByteArrayOutputStream" e "DataOutputStream", permitem fazer essa escrita em um vetor de bytes, feito isso, o ponteiro é movido para o início do arquivo e o valor do último ID utilizado é alterado, após isso, o ponteiro é movido de volta para o fim do arquivo e o registro é escrito no arquivo seguindo a seguinte ordem: Lápide(char), tamanho do registro(int) e dados(byte[]).

Feito isso, o arquivo é fechado e o novo clube está criado e registrado com sucesso, caso ocorra algum problema durante a criação, grande parte do método está dentro de um bloco "try " e caso algo dê errado cairá no bloco "catch " que interromperá a execução e mostrará uma alerta ao usuário que algo deu errado durante a criação.

2.2.2. Update

Após receber o objeto com as novas informações (porém com o mesmo ID do que será alterado) são declaradas as variáveis pos(long), lapide(char), b(byte[]), novoB(byte[]), tam(int) e objeto(Clube). Após isso, o arquivo de clubes é aberto para leitura e escrita, pulando os quatro primeiros bytes que correspondem ao último ID armazenado no arquivo, o código entra no bloco "while" onde irá ler até o último byte do arquivo. Dentro do "while", primeiramente, é armazenada a posição do ponteiro (início do registro), lápide e tamanho do arquivo nas variáveis "pos", "lapide" e "tam", respectivamente, feito isso, a variável "b" (vetor de bytes) é inicializada com o tamanho "tam" para poder armazenar os dados do registro da vez e, em seguida, os dados são lidos e armazenados.

Após armazenar os dados no vetor é feito a checagem da lápide comparando o

valor dela com "", caso seja igual, significa que o registro foi apagado e o método continua a busca pelo registro desejado no arquivo, mas caso seja diferente de "" significa que o objeto não foi deletado e o programa continua, a variável "objeto" é instanciada, então o método da classe "Clube", "fromByteArray()", que utiliza as classes "ByteArrayInputStream" e "DataInputStream" para fazer a leitura de um vetor de bytes, é chamado passando por parâmetro o vetor de bytes "b" e os dados do vetor são armazenados no objeto na sequência devida. Com o objeto armazenando os dados do vetor de bytes, é feita a checagem se o ID do objeto criado é igual ao desejado para alteração, caso não seja, o programa continua a busca para outro registro até achar o certo ou chegar ao fim do arquivo, mas caso seja o mesmo ID, é iniciado o processo de alteração.

Primeiramente, é armazenado no vetor de bytes "novoB" os dados do objeto recebido por parâmetro por meio da função "toByteArray()" explicada anteriormente no tópico do método CREATE. Após isso, é checado se o tamanho de "novoB" (objeto atualizado) é menor ou igual à variável "tam" que representa o tamanho do objeto atual no registro. Caso o tamanho do novo registro seja menor ou igual ao atual, o ponteiro é movido 6 posições a frente, ou seja, pulando tamanho do registro escrito no arquivo (não precisa ser alterado se o novo registro for menor ou igual) e o campo lápide que não deve ser alterado no método UPDATE, com o ponteiro na posição certa os dados presentes são sobrescritos pelos novos dados e o arquivo é fechado.

Caso o tamanho do novo registro seja maior que o atual, o ponteiro é movido para o final do arquivo, ocorre o processo padrão de escrita de um novo objeto (lápide, tamanho do registro e dados), o método DELETE é chamado passando por parâmetro o ID do objeto alterado (terão dois registros com o mesmo ID, porém o método DELETE apaga o primeiro que encontrar e finaliza, como o registro atualizado acabou de ser inserido no final do arquivo, o método encontrará primeiro o registro antigo e fará a exclusão da devida maneira. Após isso o arquivo é fechado e o método retorna "true", assim como o método CREATE, grande parte do método UPDATE está dentro de um bloco "try" e caso dê algum erro no processo, entrará no bloco "catch" que informará ao usuário um erro e finalizará o procedimento.

2.2.3. Delete

Recebendo por parâmetro o ID do clube que se deseja deletar é dado início ao processo de exclusão. No fim das contas, não é uma exclusão 100%, o registro é apenas marcado e sendo considerado excluído, porém ainda está presente no arquivo de modo que possa ser criada uma "lixeira", assim como nas plataformas de e-mail, mídia, arquivos, entre outras, nas quais é possível fazer a recuperação de arquivos apagados.

Dito isso, o processo inicia com a declaração das variáveis lapide(char), b(byte[]), tam(int) e objeto(Clube), abertura do arquivo para escrita e o ponteiro é movido para a posição 4 pulando os 4 primeiros bytes do arquivo que correspondem ao último ID utilizado gravado no arquivo. Após isso, o programa entra em um bloco "while" o qual, caso não tenha nenhuma parada interna, vai ler até o último byte do arquivo.

Dentro do bloco while, são lidos e armazenados os dados "externos" do registro, como, posição do ponteiro no início do registro (variável "pos"), valor do campo

lápide (variável "lapide") e tamanho do registro (variável "tam"), após guardar essas informações nas variáveis, o vetor de bytes "b" é inicializado com tamanho "tam" e, logo após, é lido do arquivo a quantidade de bytes que cabem no vetor "b", ou seja, o registro inteiro.

Depois de armazenadas todas as informações, é feita a checagem do campo lápide, se o arquivo existir (campo lápide diferente de ""), o objeto "clube" é instanciado e, por meio do método "fromByteArray()", armazena os dados do registro da vez. Feito isso, é feita a última verificação, se o ID do registro lido é igual ao inserido pelo usuário, o ponteiro é movido para a posição "pos" (armazena a posição do início do registro que corresponde ao campo lápide), sobrescreve o valor presente no campo com "", ou seja, marcando o campo lápide daquele registro e tornando-o "excluído", por fim o arquivo é fechado e o método encerrado retornando "true", ou seja, que a exclusão foi feita com sucesso.

Caso na verificação de IDs, o ID da vez não seja igual ao inserido pelo usuário, o método passa para o próximo registro até encontrar, caso chegue ao fim do arquivo e não encontre, é retornado "false", ou seja, dizendo que não foi possível apagar o registro com ID inserido, talvez pelo fato do registro não existir ou já ter sido apagado.

2.2.4. Read

No método READ temos três métodos, "readById()", "readByName()" e "readAll()", o primeiro recebe um ID por parâmetro, busca no arquivo de índices utilizando uma busca binária e retorna um valor Clube, que é o objeto que foi encontrado, o segundo recebe um nome por parâmetro e retorna o objeto caso seja encontrado, caso não exista ou tenha sido apagado, retorna "null" e, por fim, o terceiro, mostra na tela todos os clubes não deletados existentes no arquivo.

Ditas as diferenças entre eles, os processos de busca dos dois primeiros são praticamente idênticos, mudando apenas o método de busca, o arquivo é aberto para leitura, são declaradas as variáveis lapide(char), O método de busca de "readById" está explicado na seção 3, já o método de busca binária funciona da seguinte forma, b(byte[]), tam(int) e objeto(Clube). Após a declaração, o ponteiro é movido para a posição 4, pulando o valor de quatro bytes do último ID armazenado no arquivo. Então é entrado no bloco "while" que, caso não tenha nenhuma parada vinda de dentro, vai rodar pelo arquivo até o último byte, ao entrar no "while" são armazenadas as informações de lápide (variável "lapide"), tamanho do registro (variável "tam"), o vetor de bytes "b" e inicializar ele com tamanho "tam" e após isso ler do arquivo a quantidade de bytes que cabem nesse vetor e armazená-los nele. Após isso é feita a checagem da lápide para saber se o arquivo não foi deletado, caso não, a variável "objeto" é instanciada e os dados do vetor de bytes "b" são armazenados nela, por meio do método "fromByteArray()".

Após os dados serem armazenados no objeto, no caso do é feita a checagem nome(readByName()) do clube criado com os dados lidos corresponde ao nome(readByName()) recebido por parâmetro pelo método, caso seja igual, o programa retorna o objeto da classe "Clube" e fecha o arquivo, caso não, ambos os método voltam ao loop "while" até encontrar o clube desejado ou chegar ao fim do arquivo. Caso chegue

ao fim do arquivo e não encontrem o clube desejado, o arquivo é fechado e o método "readByName()" retorna "null".

No "readAll()", é quase a mesma coisa dos outros dois, com apenas uma diferença, não é feita a segunda checagem, somente a da lápide para saber se o arquivo foi excluído ou não, caso não, a variável "objeto" é instanciada o conteúdo do vetor de bytes "b" e passado para o objeto por meio do método "fromByteArray()" e os dados do objeto são impressos na tela, caso o campo lápide esteja marcado, o registro é apenas ignorado e o loop se estende até o fim do arquivo, após isso, o arquivo é fechado.

3. Busca binária

Para implementar a busca binária foi necessário percorrer o arquivo de nove em nove posições para evitar conflitos de dados que estavam dificultando a implementação. Além disso foi necessário trocar todos os métodos que precisavam do ID como parâmetro para a busca binária. O método se consiste em percorrer o arquivo de forma binária, ou seja, ordenamos o nosso arquivo de index primeiro, para então pesquisarmos os id's. Se o id for menor que o id na metade do arquivo pesquisamos do meio para trás, se for maior, pesquisamos do meio para frente. Assim que o id é encontrado lemos o arquivo até encontrar o próximo *long*, o tipo de dado utilizado para armazenar a posição, e o retornamos.

4. Lista Invertida

4.1. Implementação

Para a implementar a lista invertida foi necessário criar dois arquivos onde os dados de nome e cidade serão inseridos. As funções que implementam a lista invertida são: criar a lista, atualizar a lista, deletar a lista.



Figure 2. Estrutura da Lista Invertida

4.1.1. Criar id na lista

Para criar a lista seria necessário saber se a palavra se encontra na lista, para poder adicionar apenas o id, senão dese-se adicionar a somente o id na frente da palavra. Para estabelecer isso na frente de uma palavra seria possível adicionar apenas 5 ids e a ultima posição estaria disponível para o endereço da proxima posição da palavra onde iria armazenar mais arrays. Foi escolhida essa implementação por facilitar a questão de leitura do arquivo, pois era necessário saber quantos ids deveriam ser lidos para chegar a próxima palavra.

4.1.2. Deletar id da lista

Para deletar um id na lista o algoritmo utilizado foi ler a palavra e começar a comparar o id, se caso o read byte for igual ao id o programa deve ir para a posicao onde se encontra o id e sobrepôr com um -1, pois na lista invertida todos os dados com -1 significa que eles aquela posição está disponível para ser usada. Com isso, o id é excluído do arquivo da lista invertida.

4.1.3. Atualizar id na lista

Para atualizar a lista invertida o algoritmo utilizado foi excluir todos os dados relacionado ao id que foi atualizado, depois criar ele novamente no arquivo da lista para manter a ordenalidade e por questões de reutilização de código.

5. Arquivo de índices

Para a criação do arquivo de índices foi levado em consideração os ids e o endereço do objeto no arquivo de dados principal, então a listagem no arquivo seguiu o seguinte parâmetro (id, endereço). Para assim facilitar a busca no arquivo de índices.



Figure 3. Estrutura do arquivo de índices

5.1. Criar endereço do id no arquivo de índices

Para criar o id no arquivo de índices foi levado em consideração o tamanho do arquivo, caso o arquivo tivesse tamanho 0 o objeto ia ser inserido na primeira posição, senão ele ia ser inserido na última posição do arquivo.

5.2. Deletar id e endereço do arquivo de índices

O método de deletar levou em consideração a busca por id, quando você queria deletar um objeto do arquivo, você passava o id desse objeto e fazia a busca do mesmo. Caso achasse o id você sobre escrevia o id e o endereço com o valor -1, como se fosse uma lápide dizendo que aquele local está inutilizável.

5.3. Atualizar endereço no arquivo de índices

O método de atualizar o arquivo de índices era utilizado caso o usuário fizesse alguma alteração em um clube que demandasse a mudança de localização dele do arquivo de dados original, com isso a implementação foi feita mandando o objeto do clube onde continha o id e o endereço novo do objeto, tendo isso o algoritmo fazia uma busca pelo id e quando o achava mudava o endereço que estava escrito no seu lado para o novo endereço no arquivo de dados original.

6. Testes e resultados do arquivo sequencial

6.1. Criação de um clube

O primeiro passo para a criação do clube é escolher a opção. Após a seleção das informações seria necessário informar os dados para o programa, no exemplo forma criados dois times, com o intuito de mostrar a criação de partidas no próximo exemplo.

```
Nome: Clube Atlético Mineiro
CNPJ: 82.581.353/0001-97
Cidade: Belo Horizonte - MG
```

Figure 4. Criação clube 1

```
Nome: Cruzeiro Esporte Clube
CNPJ: 40.266.708/0001-03
Cidade: Belo Horizonte - MG
```

Figure 5. Criação clube 2

Quando criados todos os dados são inseridos no arquivo, com isso pode-se observar que todos os dados foram inseridos com sucesso.

```
00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F Decoded Text
00000000 00 00 00 04 00 20 00 00 00 33 00 00 17 43 6C 75 . . . . . 3 . . . Clu
00000010 62 65 20 41 74 6C C3 A9 74 69 63 6F 20 4D 69 6E be AtlÃ©tico Min
00000020 65 69 72 6F 00 05 31 32 33 34 35 00 0E 42 65 6C eiro . . 1 2 3 4 5 . . Bel
00000030 6F 20 48 6F 72 69 7A 6F 6E 74 65 01 01 00 20 00 o Horizonte . . . .
00000040 00 00 32 01 00 16 43 72 75 7A 65 69 72 6F 20 45 . . 2 . . . Cruzeiro E
00000050 73 70 6F 72 74 65 20 43 6C 75 62 65 00 05 35 34 sporte Clube . . 5 4
00000060 33 32 31 00 0E 42 65 6C 6F 20 48 6F 72 69 7A 6F 3 2 1 . . Belo Horiz
00000070 6E 74 65 01 00 00 20 00 00 00 28 02 00 0C 46 6C nte . . . . ( . . . Fl
00000080 61 6D 65 6E 67 6F 20 45 2E 43 00 05 36 37 38 39 amengo E.C . . 6 7 8 9
00000090 30 00 0E 52 69 6F 20 64 65 20 4A 61 6E 65 69 72 0 . . Rio de Janeir
000000A0 6F 02 04 00 20 00 00 00 45 03 00 17 43 6C 75 62 o . . . . E . . Club
000000B0 65 20 C3 81 74 6C 65 74 69 63 6F 20 4D 69 6E 65 e Ã©tletico Mine
000000C0 69 72 6F 00 12 38 32 2E 35 38 31 2E 33 35 33 2F iro . . 8 2 . 5 8 1 . 3 5 3 /
000000D0 30 30 30 31 2D 39 37 00 13 42 65 6C 6F 20 48 6F 0 0 0 1 - 9 7 . . Belo Ho
000000E0 72 69 7A 6F 6E 74 65 20 2D 20 4D 47 00 00 00 20 rizonte - MG . .
000000F0 00 00 00 44 04 00 16 43 72 75 7A 65 69 72 6F 20 . . D . . Cruzeiro
00000100 45 73 70 6F 72 74 65 20 43 6C 75 62 65 00 12 34 Esporte Clube . . 4
00000110 30 2E 32 36 36 2E 37 30 38 2F 30 30 30 31 2D 30 0 . 2 6 6 . 7 0 8 / 0 0 0 1 - 0
00000120 33 00 13 42 65 6C 6F 20 48 6F 72 69 7A 6F 6E 74 3 . . Belo Horizont
00000130 65 20 2D 20 4D 47 00 00 e - MG . .
```

Figure 6. Dados criados inseridos no arquivo

6.2. Criação de partidas

Para a criação de uma partida deve-se selecionar a opção desejada. Com isso, pode inserir os dados desejados para o programa, como mostrado e explicado nos CRUDs.


```

Insira o nome do primeiro time: Clube Atlético Mineiro
Insira o nome do segundo time: Cruzeiro Esporte Clube
Insira a quantidade de gols que o primeiro time fez: 9
Insira a quantidade de gols que o segundo time fez: 2

Partida registrada e dados alterados com sucesso!

```

Figure 7. Inserção dos dados para o programa

Após todos os dados serem inseridos em terminal terá um update nos campos de pontos e partidas jogadas no arquivo para cada um dos times, como mostrado na Figura 6 do documento.

```

00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F Decoded Text
00000000 00 00 00 04 00 20 00 00 00 33 00 00 17 43 6C 75 . . . . . 3 . . . Clu
00000010 62 65 20 41 74 6C C3 A9 74 69 63 6F 20 4D 69 6E be AtlÃ©tico Min
00000020 65 69 72 6F 00 05 31 32 33 34 35 00 0E 42 65 6C eiro . . 1 2 3 4 5 . . Bel
00000030 6F 20 48 6F 72 69 7A 6F 6E 74 65 01 01 00 20 00 o Horizonte . . . .
00000040 00 00 32 01 00 16 43 72 75 7A 65 69 72 6F 20 45 . . 2 . . . Cruzeiro E
00000050 73 70 6F 72 74 65 20 43 6C 75 62 65 00 05 35 34 spor te Clube . . 5 4
00000060 33 32 31 00 0E 42 65 6C 6F 20 48 6F 72 69 7A 6F 3 2 1 . . Belo Horiz o
00000070 6E 74 65 02 00 00 20 00 00 28 02 00 0C 46 6C nte . . . . . ( . . . Fl
00000080 61 6D 65 6E 67 6F 20 45 2E 43 00 05 36 37 38 39 amengo E. C. . 6 7 8 9
00000090 30 00 0E 52 69 6F 20 64 65 20 4A 61 6E 65 69 72 0 . . Rio de Janeir
000000A0 6F 02 04 00 20 00 00 45 03 00 17 43 6C 75 62 o . . . . . E . . . Club
000000B0 65 20 C3 81 74 6C 65 74 69 63 6F 20 4D 69 6E 65 e Ã tletico Mine
000000C0 69 72 6F 00 12 38 32 2E 35 38 31 2E 33 35 33 2F iro . . 8 2 . 5 8 1 . 3 5 3 /
000000D0 30 30 30 31 2D 39 37 00 13 42 65 6C 6F 20 48 6F 0 0 0 1 - 9 7 . . Belo Ho
000000E0 72 69 7A 6F 6E 74 65 20 2D 20 4D 47 01 03 00 20 rizonte - MG . .
000000F0 00 00 00 44 04 00 16 43 72 75 7A 65 69 72 6F 20 . . D . . Cruzeiro
00000100 45 73 70 6F 72 74 65 20 43 6C 75 62 65 00 12 34 Esporte Clube . . 4
00000110 30 2E 32 36 36 2E 37 30 38 2F 30 30 30 31 2D 30 0 . 2 6 6 . 7 0 8 / 0 0 0 1 - 0
00000120 33 00 13 42 65 6C 6F 20 48 6F 72 69 7A 6F 6E 74 3 . . Belo Horizont
00000130 65 20 2D 20 4D 47 00 00 e - MG . .

```

Figure 8. Resultado no arquivo após a inserção

6.3. Procurar um time no arquivo

Para procurar um time no arquivo, após selecionar a opção desejada basta apenas inserir o ID desejado que será mostrado todas as informações disponíveis para determinado clube.

```

1 - Criar time
2 - Criar partida
3 - Procurar time
4 - Deletar time
5 - Atualizar dados do time
6 - Listar todos os times
0 - Sair

Digite sua opção: 3

Insira o ID que deseja pesquisar:
0

```

Figure 9. Seleção da opção desejada e passagem do ID

```
ID: 0
Nome clube: Clube Atlético Mineiro
CNPJ: 12345
Cidade: Belo Horizonte
Partidas jogadas: 1
Pontos acumulados: 1
```

Figure 10. Resultado da pesquisa por ID

6.4. Atualizar dados do time

Para o exemplo vamos mudar o nome de Cruzeiro Esporte Clube para Flamengo, além disso mudar o CNPJ e a cidade do clube.

```
1 - Criar time
2 - Criar partida
3 - Procurar time
4 - Deletar time
5 - Atualizar dados do time
6 - Listar todos os times
0 - Sair

Digite sua opção: 5

Insira o ID do time que deseja alterar: 1
```

Figure 11. Seleção e passagem do ID

```
ID: 1
Nome clube: Cruzeiro Esporte Clube
CNPJ: 54321
Cidade: Belo Horizonte
Partidas jogadas: 2
Pontos acumulados: 0

***Insira a seguir os novos dados***

Nome: Flamengo
CNPJ: 91.088.704/0001-13
Cidade: Rio de Janeiro - RJ
Registro atualizado com sucesso!
```

Figure 12. Novos dados inseridos

Quando fazemos a atualização pode-se ver que dependendo da quantidade de bytes a serem inseridos pode ser modificado de formas diferentes como mostrado na seção CRUD - Update. Com isso, o resultado do arquivo após a atualização está sendo mostrado na Figura 11 do documento.

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded Text
00000000	00	00	00	04	00	20	00	00	00	33	00	00	17	43	6C	75	...3...Clu
00000010	62	65	20	41	74	6C	C3	A9	74	69	63	6F	20	4D	69	6E	be AtlÃ©tico Min
00000020	65	69	72	6F	00	05	31	32	33	34	35	00	0E	42	65	6C	eiro...12345...Bel
00000030	6F	20	48	6F	72	69	7A	6F	6E	74	65	01	01	00	2A	00	o Horizonte...*
00000040	00	00	32	01	00	16	43	72	75	7A	65	69	72	6F	20	45	...2...Cruzeiro E
00000050	73	70	6F	72	74	65	20	43	6C	75	62	65	00	05	35	34	sporte Clube...54
00000060	33	32	31	00	0E	42	65	6C	6F	20	48	6F	72	69	7A	6F	321...Belo Horizo
00000070	6E	74	65	02	00	00	20	00	00	00	28	02	00	0C	46	6C	nte... (...Fl
00000080	61	6D	65	6E	67	6F	20	45	2E	43	00	05	36	37	38	39	amengo E.C...6789
00000090	30	00	0E	52	69	6F	20	64	65	20	4A	61	6E	65	69	72	0...Rio de Janeir
000000A0	6F	02	04	00	20	00	00	00	45	03	00	17	43	6C	75	62	o...E...Club
000000B0	65	20	C3	81	74	6C	65	74	69	63	6F	20	4D	69	6E	65	e Ãtletico Mine
000000C0	69	72	6F	00	12	38	32	2E	35	38	31	2E	33	35	33	2F	iro...82.581.353/
000000D0	30	30	30	31	2D	39	37	00	13	42	65	6C	6F	20	48	6F	0001-97...Belo Ho
000000E0	72	69	7A	6F	6E	74	65	20	2D	20	4D	47	01	03	00	20	rizonte - MG...
000000F0	00	00	00	44	04	00	16	43	72	75	7A	65	69	72	6F	20	...D...Cruzeiro
00000100	45	73	70	6F	72	74	65	20	43	6C	75	62	65	00	12	34	Esporte Clube...4
00000110	30	2E	32	36	36	2E	37	30	38	2F	30	30	30	31	2D	30	0.266.708/0001-0
00000120	33	00	13	42	65	6C	6F	20	48	6F	72	69	7A	6F	6E	74	3...Belo Horizont
00000130	65	20	2D	20	4D	47	00	00	00	20	00	00	00	36	01	00	e - MG...6...

Figure 13. Resultado no arquivo

6.5. Deletar clube

Ao deletar um clube uma lápide é inserida no arquivo.

Insira o ID do usuário que deseja deletar:
2
Time deletado com sucesso!

Figure 14. Passagem dados

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded Text
00000000	00	00	00	04	00	20	00	00	00	33	00	00	17	43	6C	75	...3...Clu
00000010	62	65	20	41	74	6C	C3	A9	74	69	63	6F	20	4D	69	6E	be AtlÃ©tico Min
00000020	65	69	72	6F	00	05	31	32	33	34	35	00	0E	42	65	6C	eiro...12345...Bel
00000030	6F	20	48	6F	72	69	7A	6F	6E	74	65	01	01	00	2A	00	o Horizonte...*
00000040	00	00	32	01	00	16	43	72	75	7A	65	69	72	6F	20	45	...2...Cruzeiro E
00000050	73	70	6F	72	74	65	20	43	6C	75	62	65	00	05	35	34	sporte Clube...54
00000060	33	32	31	00	0E	42	65	6C	6F	20	48	6F	72	69	7A	6F	321...Belo Horizo
00000070	6E	74	65	02	00	00	20	00	00	00	28	02	00	0C	46	6C	nte... (...Fl
00000080	61	6D	65	6E	67	6F	20	45	2E	43	00	05	36	37	38	39	amengo E.C...6789
00000090	30	00	0E	52	69	6F	20	64	65	20	4A	61	6E	65	69	72	0...Rio de Janeir
000000A0	6F	02	04	00	20	00	00	00	45	03	00	17	43	6C	75	62	o...E...Club
000000B0	65	20	C3	81	74	6C	65	74	69	63	6F	20	4D	69	6E	65	e Ãtletico Mine
000000C0	69	72	6F	00	12	38	32	2E	35	38	31	2E	33	35	33	2F	iro...82.581.353/
000000D0	30	30	30	31	2D	39	37	00	13	42	65	6C	6F	20	48	6F	0001-97...Belo Ho
000000E0	72	69	7A	6F	6E	74	65	20	2D	20	4D	47	01	03	00	20	rizonte - MG...
000000F0	00	00	00	44	04	00	16	43	72	75	7A	65	69	72	6F	20	...D...Cruzeiro
00000100	45	73	70	6F	72	74	65	20	43	6C	75	62	65	00	12	34	Esporte Clube...4
00000110	30	2E	32	36	36	2E	37	30	38	2F	30	30	30	31	2D	30	0.266.708/0001-0
00000120	33	00	13	42	65	6C	6F	20	48	6F	72	69	7A	6F	6E	74	3...Belo Horizont
00000130	65	20	2D	20	4D	47	00	00	00	20	00	00	00	36	01	00	e - MG...6...

Figure 15. Resultado no arquivo após deletar clube

6.6. Buscar ID's na lista invertida

Para o teste vamos criar 4 times: Clube Atlético Mineiro (ID = 0, Belo Horizonte), Clube Regatas do Flamengo (ID = 1, Rio de Janeiro), Cruzeiro Esporte Clube (ID = 2, Belo Horizonte) e Vasco da Gama (ID = 3, Rio de Janeiro). E para testar o algoritmo de busca na lista invertida vamos fazer duas buscar por nome e duas buscas por cidade, utilizando as seguintes informações Clube e Rio, respectivamente.

6.6.1. Busca feita por nome

```
Digite sua opção: 7
Deseja busca por nome(1) ou cidade(2)
1
Digite o nome que deseja:
Clube
Os ID's relacionados a palavra digitada foi:
[0, 1, 2]
```

Figure 16. Resultado da busca feita por nome na lista invertida

6.6.2. Busca feita por cidade

```
Digite sua opção: 7
Deseja busca por nome(1) ou cidade(2)
2
Digite a cidade que deseja:
Rio
Os ID's relacionados a palavra digitada foi:
[1, 3]
```

Figure 17. Resultado da busca feita por cidade na lista invertida

6.7. Arquivo de índices

O teste no arquivo de índices foi feito a partir de uma função específica não disponível para o usuário, apenas para o programador testar. Então para o teste foi efetuado a criação de 4 times (os mesmos criados no buscar id's na lista invertida e foram deletados, atualizados para ver se a alteração acontecia no arquivo de índices.

```
Digite sua opção: 8
ID.....: 0 ENDEREÇO: 4
ID.....: 1 ENDEREÇO: 59
ID.....: 2 ENDEREÇO: 117
ID.....: 3 ENDEREÇO: 172
```

Figure 18. Arquivo de índices antes da modificação

6.7.1. Deletar no arquivo de índices

Após deletar o id e endereço do arquivo o resultado esperado e não mostrar mais para o programador quando *printar* o arquivo inteiro.

```
Digite sua opção: 8
ID.....: 0 ENDEREÇO: 4
ID.....: 2 ENDEREÇO: 117
ID.....: 3 ENDEREÇO: 172
```

Figure 19. Arquivo de índices depois do delete

6.7.2. Atualizar no arquivo de índices

Após atualizar o id e endereço do arquivo o resultado esperado e mostrar para o programador a nova posição do id atualizado.

```
Digite sua opção: 8
ID.....: 0 ENDEREÇO: 4
ID.....: 2 ENDEREÇO: 218
ID.....: 3 ENDEREÇO: 172
```

Figure 20. Arquivo de índices depois de atualizar o ID 2

7. Ordenação externa

Infelizmente o grupo não conseguiu implementar de forma correta a ordenação externa. Para não prejudicar o restante do trabalho optamos por armazenar todos os id's no arquivo de índices já de forma ordenada, para possibilitar a implementação de outros requisitos impostos no trabalho.

8. Conclusão

De acordo com o resultado dos testes, percebemos que o programa está funcionando de forma correta, conseguindo cobrir todos os tipos de situações que podem ser causadas por dados inseridos pelo usuário sendo eles válidos ou não. Além disso pode-se observar a grande importância da utilização do arquivo de índices juntamente da substituição da busca sequencial por uma busca binária no arquivo de índices que estava ordenado utilizando uma ordenação na inserção dos id's, melhorando assim a performance da busca. O programa também utilizou uma lista invertida para facilitar a busca do usuário quando era necessário busca os id's relacionados a determinada palavra (nome e/ou cidade). Pode-se observar então que o programa está em pleno funcionamento com muitas opções para facilitar a interação do usuário.