Documentação - Trabalho Prático 3
Algoritmos e Estrutura de Dados 3

Lucas Carvalho da Luz
Vinícius Henrique Giovanini

[1]ICEI – Pontifícia Universidade Católica de Minas Gerais (PUC)
Belo Horizonte – MG – Brazil

{lcluz,vgiovanini}@sga.pucminas.br

Resumo. *Após a implementação de um CRUD em arquivo sequencial desenvolvido no TP1, e o desenvolvimento do arquivo de dados e da lista invertida no TP2, o TP3 foi incluindo uma compressão e criptografia nos arquivos de dados.*

1. Introdução

Primeiramente, a diferença entre um arquivo sequencial e um indexado é que no sequencial as informações são armazenadas em ordem de inserção, logo, ao realizar uma busca em um arquivo sequencial observamos uma busca de **registro por registro**, em um arquivo indexado os registros se encontram em um arquivo diferente do arquivo onde a pesquisa é realizada, um arquivo indexado possui um segundo arquivo auxiliar onde são armazenados identificadores e endereços, tornando a pesquisa muito mais rápida, além de possibilitar uma busca não linear, no caso foi implementada a pesquisa binária que é 2X mais rápida.

A criptografia é utilizada desde tempos antigos, e foi muito utilizada nas grandes guerras, quando foram criadas algumas máquinas para ser repassado mensagens, hoje em dia com o grande uso da internet a criptografia é utilizada principalmente para ”esconder” mensagens enviadas, e só descriptografando quando o usuário for visualizar. E com a interação na internet sendo muito utilizada veio a necessidade da compressão de dados, para que os mesmos sejam transferidos pela internet de maneira mais eficiente.

2. Desenvolvimento

O trabalho prático 2, foi uma extensão do TP01, mas a proposta do TP02 era utilizar um sistema de arquivos de índices, e nele fazer a pesquisa do registro desejado através da pesquisa binária, mas para essa pesquisa ser executada os arquivos tem que estar ordenados, dessa forma foi implementado a ordenação externa. O último tópico proposto foi a implementação da lista invertida, na qual foi implementada para fazer a busca por String do nome do time e da cidade do time, e quando for pesquisar pelo ID, ele utiliza o arquivo de índice.

A proposta do trabalho prático 3 era inserir a criptografia (Escolha) e a compressão (LZW) no arquivo de dados, dessa forma foi implementado a criptografia da tabela de **Vigenère** que é uma cifra de substituição através de chave e a compressão foi utilizando o algoritmo **LZW**

Dessa forma a classe chamada futebolprincipal é a classe main do programa, a classe arquivocrud contém os métodos para o CRUD em ambos os três arquivos de dados, que são eles futebol.db, na qual armazena o registro inteiro dos clubes, aindices.db responsável pelo registro do índice, e o listainvertida.db armazenando os nomes dos times e as cidade para ser pesquisado, foi criado a classe criptografia presente os métodos para criar a cifra, e a classe compressão com os métodos referentes ao algoritmo LZW. As demais classes foram classe para apoiar as diversas partes do programa, como a classe ordenacaoexterna que auxilia a pesquisa no arquivo de índice, e a classe índice auxilia as operações no arquivo de índice, e essas foram as classes implementadas:

- futebolprincipal
- arquivocrud
- fut
- indice
- listainvertida
- ordenacaoexterna
- compressao
- criptografia

2.1. Método de Escrita - Create

A escrita dos registros foi feito através da classe arquivocrud, com a ajuda de dois métodos, a função **criarClube**, na qual recebe os dados do novo clube que será registrado, e manda esses dados para o método **escreverArquivo**, na qual é responsável pela a escrita nos três arquivos, no arquivo principal, no arquivo de índice, e na lista invertida.

A escrita no arquivo principal, nomeado como futebol.db é realizada com uma criptografia no **Nome**, presente no método escreverArquivo.

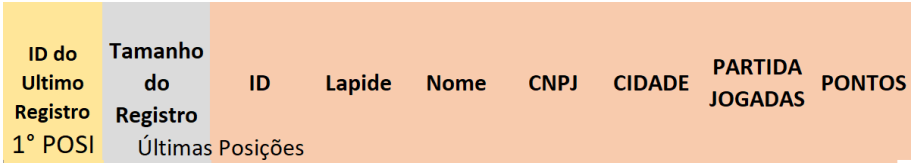


Figure 1. Ordem de escrita dos bytes: Amarelo: ID começo do arquivo. Cinza: Tamanho do Registro. Laranja: Array de Bytes.

A escrita no arquivo de índice é realizada salvando três dados na lista, uma variável short contendo o ID, um long com a posição do registro no arquivo de dados principal (futebol.db), e logo após uma string que é a lápide, somando 13 bytes por registro no arquivo de índice (aindices.db).

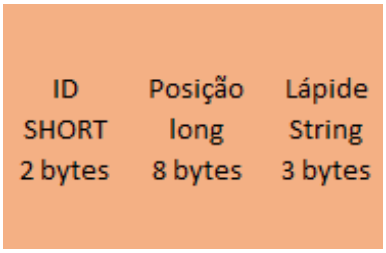


Figure 2. Ordem de escrita dos bytes no arquivo de índice (aindices.db)

Durante a escrita inicial no arquivo de índice, o método presente na classe índice que está encarregado de fazer a escrita está fazendo também um embaralhamento dos registros antes de fazer a escrita propriamente. O embaralhamento consiste basicamente em salvar os índices pares antes dos impares, lembrando que no arquivo de índice a busca é feita pelo ID, sem considerar o ID zero. Dessa forma foi realizado algumas verificações, pois basicamente o método salva números impares pulando 13 bytes, e pares no bytes pulados, deixando o arquivo mais embaralhado, porém logo após que é feito a primeira ordenação do arquivo de índice os próximos elementos inseridos não contém esse embaralhamento.

Figure 3. Na parte de cima, está um exemplo do salvamento dos índices sem embaralhamento, na de baixo com embaralhamento

A escrita no arquivo da Lista Invertida é realizada pelo método escreverArquivo, dessa forma ele salva o nome (String) que corresponde ao nome do próprio clube ou a cidade do mesmo, e caso na hora de inserir os dados não seja especificado a cidade, não é implementado na lista, caso seja informado a cidade será sempre feita duas escritas na LI, que seria referente ao nome e a cidade inserida, essa escrita irá gastar sempre o número de bytes referentes a String inserida, que será de tamanho variável e logo após a posição desse registro no arquivo de dados principal (futebol.db) referente a 8 bytes do long. A escrita do campo nome é feito com o mesmo criptografado.

2.2. Método de Pesquisa - Read

Para realizar uma pesquisa com mais eficiência, foi utilizado os arquivos indexados para fazer a busca pelo ID, e a lista invertida para fazer a busca por nome e cidade, que contém o posicionamento de cada registro no arquivo de dados principal, porém a pesquisa no arquivo de índice foi feito de maneira diferente do que na lista invertida.

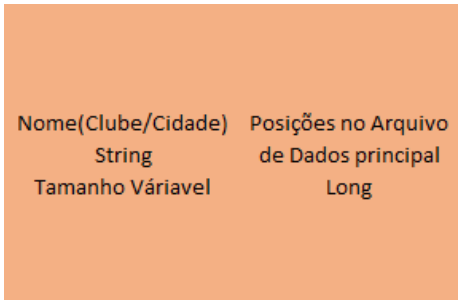


Figure 4. Ordem de escrita dos bytes no arquivo da Lista Invertida (listainvertida.db)

No arquivo indexado foi utilizado uma busca binária, porém para que ela seja executada de maneira correta os elementos presentes tem que estar ordenados, dessa forma utiliza-se a distribuição e a ordenação externa para fazer a ordenação desses registros, e a busca binária é encarregada de identifica-los. Mas antes de fazer a ordenação externa existe um método da classe arquivocrud **temMargemZero** que identifica caso o arquivo esteja com pulos de 13 bytes, referente ao embaralhamento da escrita, dessa maneira para fazer essa correção e retirar esses pulos dentro do arquivo o método corrigirArquivoIndice da classe ordenacaoexterna salva todos os bytes do arquivo sem pegar os bytes vazios, e manda para o método limpaArquivoDozeros da mesma classe, na qual reescreve o arquivo sem os pulos de 13 bytes do zero.

A busca na Lista Invertida é utilizada para procurar pelo nome do clube e pela cidade através da classe listainvertida e do método pesquisaListaInvertida, dessa forma ela faz a leitura sequencialmente de uma String e compara se é essa a String procurada, caso não seja, ela pula **8 bytes X 20**, pois cada Nome na lista tem no máximo 20 espaços para guardar endereços com o mesmo nome, quando acha a String ele vai lendo long por long e concatenando em uma string dividida por ponto e vírgula, e caso seja encontrado um long escrito **-10** consiste que a lista chegou ao fim, logo após manda a String concatenada para o método desmembrarStringListaInvertida, para pegar essa string e desmembrar em um array de long, que logo em seguida é mandado para o método imprimirListaInvertida, que pega todas as posições presente no array e faz a leitura no arquivo de dados principal.

Antes de realizar uma busca na Lista Invertida, foi implementado uma verificação, dessa maneira é necessário informar se é uma pesquisa referente ao clube ou a uma cidade, pois caso seja pesquisado um clube o nome que está sendo pesquisado tem que ser criptogrado para ser encontrado com o nome criptografado na lista, e caso seja uma cidade tem que pesquisar sem essa criptografia, pois cidade não é um campo cifrado.

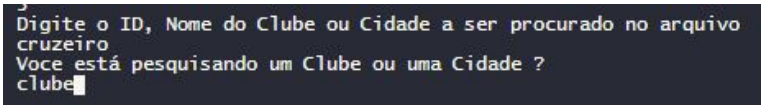


Figure 5. Verificação do tipo de Pesquisa

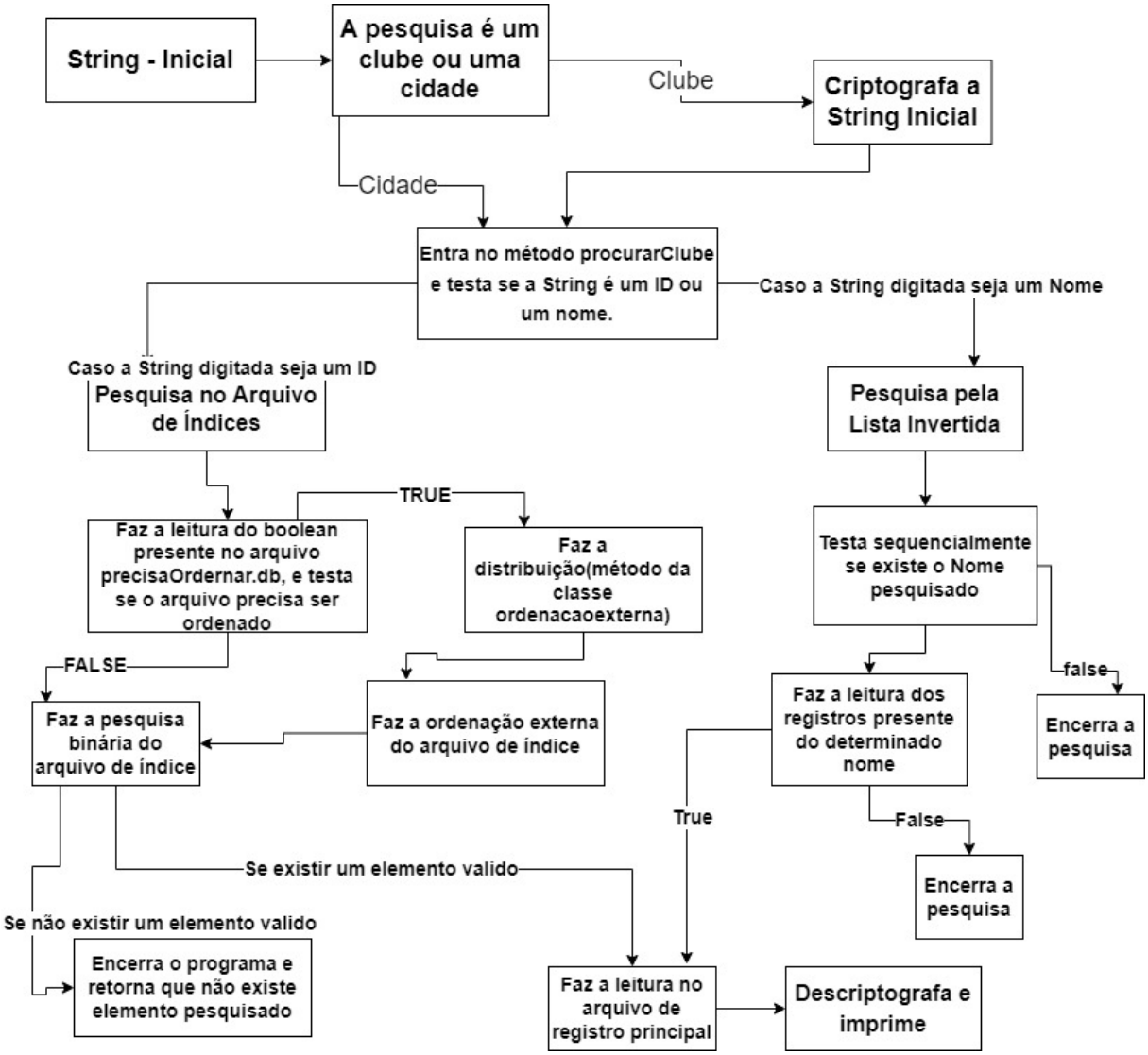


Figure 6. Diagrama de decisões do método de pesquisa

2.3. Método de Delete

O delete dos registros é feito pelo método presente na classe arquivoCrud chamado arquivoDelete, através do índice informado, nele é feito o delete dos elementos nos três arquivos de dados de maneira diferente.

No arquivo principal, o delete é feito fazendo uma pesquisa do elemento no arquivo de índice, pegando sua posição e marcando a lápide no arquivo principal, também no arquivo de índice é feito a marcação da lápide após a busca. Mas na lista invertida é feito a busca pelo nome do elemento que é pego quando vai marcar a lápide no arquivo principal, e quando acha o elemento o método **arquivoDeleteNalIstaInvertida** procura pela posição do elemento na LI, e quando acha ele não marca a lápide, ele sobrescreve com zero a posição do registro que quer ser apagado, dessa forma quando for fazer a

pesquisa na LI pelo nome e caso agora tenha apagado o único registro presente para esse nome, não será impresso nada, pois tem o nome na LI mas não tem nenhuma posição salva nesse nome na lista invertida.

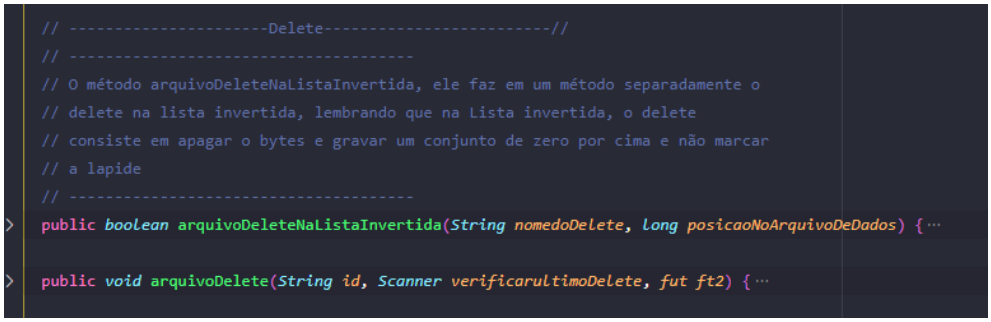


Figure 7. Métodos para realizar o Delete dos três arquivos de dados!

2.4. Método de Update

O método Update implementado pode ser completo ou incompleto, quando ele for completo ele irá alterar todos os campos do clube inserido pelo próprio usuário, e quando ele for incompleto ele irá alterar somente a partida e os pontos do clube automaticamente, dessa forma quando o método for incompleto ele não irá precisar mudar de posição no registro principal, pois estará mudando sempre dois números bytes, assim quando for feito esse tipo de Update, os arquivos da lista invertida e de índices não são alterados, porém quando a alteração é completa é feita o Update tanto no arquivo principal, quanto no arquivo de índice mudando somente o valor do long no registro, e também na lista invertida, encontrando o valor antigo e sobrescrevendo ele pelo novo, alterando assim o registro em todos os arquivos de armazenamento.

O registro a ser atualizado é sempre criptografado para fazer a busca, e quando for inserido o nome atualizado ele é criptografado e adicionado no arquivo principal e nos arquivos da lista e índice

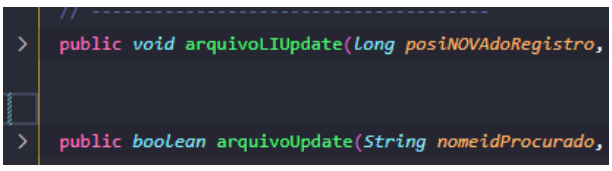


Figure 8. Métodos para realizar o Update dos três arquivos de dados!

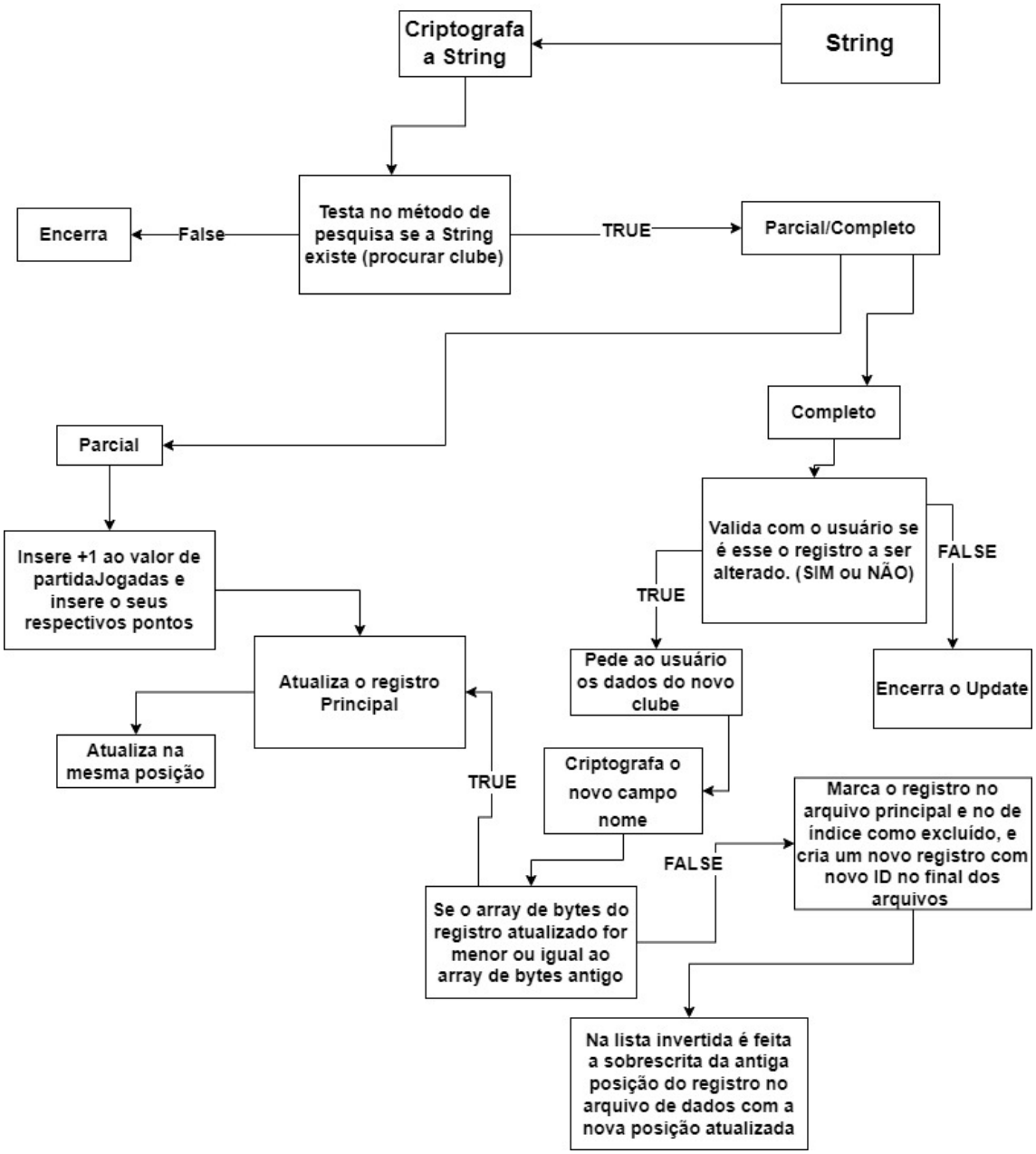


Figure 9. Diagrama de decisões do método de Update

2.5. Método Realizar partida

O método **realizarPartida** presente na classe futebolprincipal, realiza um confronto entre dois times, que são definidos pelo seu ID, buscando assim pelo método de buscar por índice, logo após é necessário confirmar se esses são os clubes corretos para a partida e assim insere o placar, resultando no incremento de mais um ao número de partida jogadas a ambos os clubes e fazendo o incremento ao número de pontos referente a vitória, derrota ou empate respectivamente.

```
public static boolean realizarPartida(Scanner entrada) {

  // Esse método retorna true e false para saber se a partida foi relizada com
  // sucesso, ele recebe o nome dos 2 time pede o placar e de acordo com o
  // resultado da partida chama o método arquivoUpdate para atualizar ambos os
  // times no arquivo. Erros foram tratados, caso digite um clube que não exista !
```

Figure 10. Cabeçalho do método realizarPartida

3. Criptografia e Descriptografia

A criptografia escolhida foi a Cifra de Vigenère, que consiste em uma cifra de substituição, com utilização de uma **chave**, que no trabalho prático e usada como padrão a palavra **AEDS**, dessa maneira a cifra utiliza a chave e compara com a palavra a ser criptografada, e assim é pesquisado na linha a primeira letra da palavra a ser criptografada com a primeira letra da chave porem com a coluna, resultando assim na letra a ser substituída.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

Figure 11. Tabela da Cifra de Vigenère

O campo criptografado é o campo Nome, dessa maneira é realizado a criptografia na escrita inicial do registro, salvando o nome criptografado no arquivo principal e nos arquivos de índice e lista invertida, é realizada a criptografia na escrita também no update, quando é necessário salvar um ”novo” registro, a criptografia também está presente no método read, pois ele criptografa a entrada do usuário para pesquisar na base de dados, criptografada.

A descriptografia acontece sempre que o método read for chamado, assim ele faz a pesquisa e quando vai imprimir ele descriptografa o campo nome e mostra ele com seu verdadeiro ”valor”, utilizando a mesma cifra de Vegenère porém ele pesquisa a letra criptografa, e após isso pesquisa qual é a letra da linha que corresponde a esse caractere criptografado na tabela, descobrindo seu verdadeiro valor, fazendo letra por letra até conseguir a String descriptografada.

Quando o método de criptografia tenta realizar a cifra em uma String que contém caracteres não presente na tabela ele retorna uma String vazia, assim os métodos de escrita salvam o valor sem criptografia e a busca realiza uma leitura e não descriptografa depois.

4. Compressão e Descompressão

A compressão e descompressão foram feitas utilizando o algoritmo **LZW**, através da classe compressao foram implementados métodos para realizar a compressão do arquivo de dados principal, nos campos nome, cnpj e cidade, e quando os últimos dois campos não possuírem dados validados ele não realiza a compressão e salva o dado de maneira igual ao arquivo de dados principal, porém antes de cada campo compressado ele salva um byte boolean, que se tiver **false** consiste que o campo não foi compressado, e se tiver **true** consiste que o campo foi compressado, e no final do campo é escrito um byte **-10**, que consiste o final do array de bytes compressado.

O byte comprimido é salvo em um novo arquivo numerado pelo usuário, na pasta database -¿ compressão, assim para cada compressão é criado 3 arquivos de dados, o primeiro contém o byte array comprimido, e os outros dois são a lista invertida e o arquivo de índice salvos naquele momento.

A compressão também possui um método que faz o calculo da melhora no tamanho do byte array comparado com o arquivo inicial, resultando em uma porcentagem de compressão.

A descompressão foi implementada na classe compressao, e ela descomprime e salva no arquivo de dados principal, além de salvar os dados do índice e da lista referente a compressão nos arquivos principal do mesmos, deixando assim o arquivo descomprimido pronto para ser utilizado os métodos do CRUD.

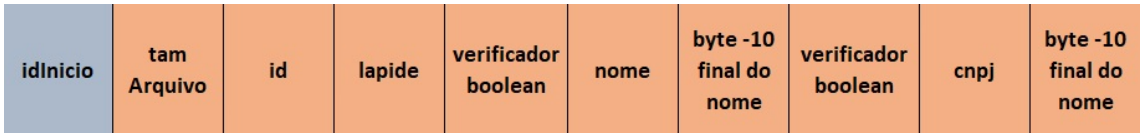
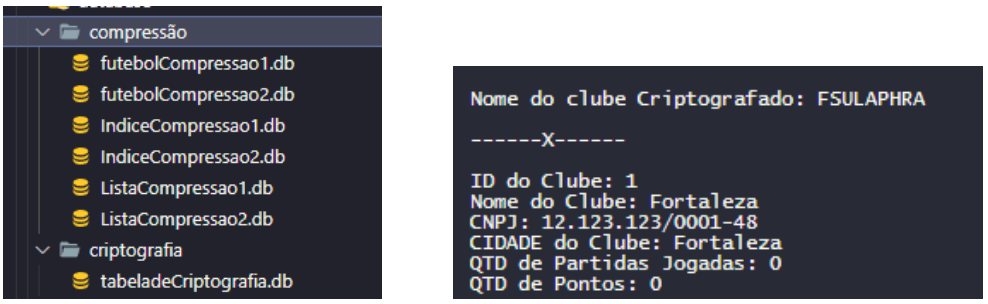


Figure 12. Byte array da compressao de 1 registro (Ilustrado até o CNPJ, falto Cidade Partida Jogadas e Pontos)

5. Testes e Resultados

A criptografia foi realizada nos métodos que realizam alguma escrita em algum arquivo de dado, e na entrada do usuário quando se procura algum registro, a descriptografia foi implementada nos métodos de impressão, resultando em uma escrita e pesquisa correta.

A compressão realiza o lzw corretamente, além de efetuar um calculo de melhoria da compressão, salvando esses dados em um novo arquivo de dados, permitindo que a descompressão volte para o arquivo principal esses dados e que eles possam voltar a serem manipulados pelos métodos do CRUD.



(a) Arquivos de Compressao e a Tabela de Vinegère

(b) Clube Criptografado

Figure 13. Imagens da classe Compressao e Criptografia

6. Conclusão

Pode-se concluir com a adição do arquivo de índice, e com a ordenação, a busca fica completamente mais otimizada, pois no arquivo indexado possui uma forma resumida dos bytes do registro, e quando se pesquisa pode-se utilizar métodos como a busca binária melhorando o tempo de resposta caso tenha um arquivo com muito registros, e para dar suporte a busca binária foi realizado a ordenação externa, que é um método para ordenar arquivos em disco, através de caminhos, nesse TP02 foi utilizado uma OE de caminho de tamanho 10, podendo ordenar 20 elementos por vez, e logo depois a busca binária localizando o registro requerido.

Pode-se constatar que a implementação da lista invertida é muito importante, pois ela abre a chance de retornar mais de um registro por pesquisa, caso o time tenha o mesmo nome, e caso os times sejam da mesma cidade, assim podendo visualizar mais registros através de uma única pesquisa.

Pode-se concluir que apesar que com os impactos significativo na performance e no uso de memoria da aplicação, a adição das funções da cripto e compre desempenho um papel impor na otimização de espaço em disco e também na segurança e privacidade dos dados, sem comprometer a robustez e expansibilidade.

7. Referências

Os conteudos usados para a criação do trabalho foi as vídeos aulas de Algoritmo e Estrutura de Dados 3 [Kutova 2022], além das aulas presencias [Soares 2022], e o documento teve base no código criado que está presente no GitHub [Giovanini 2022]

References

Giovanini, V. H. (2022). Repositório do github de vinícius h. `https://github.com/viniciushgiovanini/CRUD-Futebol-TP02-AEDS3`. Accessed: 2022-05-15.

Kutova, M. (2022). Aulas online do curso de algoritmo e estrutura de dados 3. `https://www.pucminas.br`. Accessed: 2022-05-15.

Soares, F. (2022). Aulas do curso de algoritmo e estrutura de dados 3. `https://www.pucminas.br`. Accessed: 2022-05-15.