Trabalho Prático de Algoritmos e Estruturas de Dados 3

Amanda Canizela Guimarães[1], **Ariel Inácio**[2]

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

Abstract. *This article presents all the information on the development of the work for the subject 'Algoritmos e Estruturas de Dados 3' of the Computer Engineering course. During this strict, it will be presented all the tests, evolution, and choices that were made by the pair during all the steps of the project.*

Resumo. *Este artigo apresenta as informações de desenvolvimento do trabalho prático da matéria de Algoritmos e Estruturas de Dados 3 do curso de Engenharia de Computação. Ao longo deste, apresentam-se os testes, os desenvolvimentos e as escolhas optadas pela dupla durante a resolução de todos os passos do projeto.*

1. Introdução

Este artigo documenta o desenvolvimento dos trabalhos práticos (TPs) da disciplina de 'Algoritmos e Estruturas de Dados 3'. O projeto foi redigido com o objetivo de organizar e apresentar todas as etapas de confecção dos trabalhos realizados ao longo do semestre, desde os códigos, os erros e dificuldades até alguns detalhes sobre os vídeos feitos pela dupla.

Ao longo do texto, são detalhados os desafios e as soluções adotadas para cada 'TP', passando pelas explicações dos trabalhos de 1 (um) ao 4 (quatro). Em cada seção, são apresentados os testes, as melhorias e a implementação, proporcionando uma visão completa do projeto.

2. Desenvolvimento

Este trabalho tem como objetivo explicar detalhadamente o processo de criação e de desenvolvimento do trabalho desenvolvido ao longo de todo o semestre. Para garantir uma completa explicação e de fácil entendimento, o tópico em questão foi dividido em quatro seções, as quais abordam individualmente cada parcela do projeto.

Inicialmente, será apresentada uma explicação do TP1, o qual foi composto pela criação da base de dados; pela manipulação de arquivo sequencial e pela ordenação externa. Neste, será explicada sua implementação e os desafios de seu desenrolar. Na sequência, explica-se o TP2 e suas demandas, as quais implicam na manipulação de arquivo indexado com árvore B/B+/'B estrela', de hash e de lista invertida. Assim como a subseção anterior, as etapas e os processos abordados serão os mesmos. Por fim, a mesma dinâmica será trazida para os TPs 3 e 4, os quais tratarão de explicar compactação e casamento de padrões em um e criptografia em outro, respectivamente.

2.1. TP1

Neste primeiro trabalho prático fizemos uma implementação de um arquivo CRUD sequencial com base em um banco de dados da Netflix na qual contém séries e filmes presentes em seu catálogo, seguindo essas regras de entidades no registro:

- String de tamanho fixo = Nome dos países com a abreviação de 3 letras (ISO 3166-1 alfa-3)
- String de tamanho variável = Nome dos Filmes/Series assim como o nome dos diretores
- Data = Data de adição ao catalogo
- Lista de valores com separador = Lista de gêneros do Filme/Serie
- Inteiro ou Float = Inteiro destinado a indicar o ID

2.2. Criação da classe

Criamos uma classe chamada ”Filmes”, utilizando Externalizable, para organizar tratar, escrever e ler os dados, tornando a criação do código mais dinâmica e simples.

Código JAVA

```
1
2  public class Filmes implements Externalizable, Comparable<Filmes
       >{
3
4      private boolean lapide;
5      private int id;
6      private String tipo;
7      private String nome;
8      private String diretor;
9      private String pais;
10     private LocalDate ano_adi;
11     private Year ano_lan;
12     private String classificacao;
13     private String duracao;
14     private String Genero;
15
16     public Filmes(){}
17
18     public Filmes(List<String> lista, int tmp, Boolean
           formasFormatacao) {
19         // Define o formato da data com base na  o p   o
               foramasFormatacao que e fornecia de acordo com a
               funcao que que instancia esta
20         DateTimeFormatter format;
21         if (formasFormatacao) {
22             format = DateTimeFormatter.ofPattern("yyyy/M/d");
23             LocalDate data = LocalDate.parse(lista.get(5),
                   format);
24             format = DateTimeFormatter.ofPattern("M/d/yyyy");
25             lista.set(5, data.format(format));
```

```
26          } else {
27              format = DateTimeFormatter.ofPattern("M/d/yyyy");
28          }
29
30          // Inicializa os atributos do objeto Filmes
31          this.lapide = false;
32          this.id = tmp;
33          this.tipo = lista.get(1);
34          this.nome = lista.get(2);
35          this.diretor = lista.get(3);
36          this.pais = lista.get(4);
37
38          // Converte a data para LocalDate
39          LocalDate data = LocalDate.parse(lista.get(5), format);
40          this.ano_adi = data;
41
42          // Converte o ano de lan amento para Year
43          Year anoLan = Year.parse(lista.get(6));
44          this.ano_lan = anoLan;
45
46          this.classificacao = lista.get(7);
47          this.duracao = lista.get(8);
48          this.Genero = lista.get(9);
49      }
50
51      @Override
52      public void writeExternal(ObjectOutput Out) throws
            IOException {
53          // Escreve os atributos no fluxo de sa da
54          Out.writeBoolean(lapide);
55          Out.writeInt(id);
56
57          byte[] tipoBytes = tipo.getBytes("UTF-8");
58          Out.writeShort(tipoBytes.length);
59          Out.write(tipoBytes);
60
61          byte[] nomeBytes = nome.getBytes("UTF-8");
62          Out.writeShort(nomeBytes.length);
63          Out.write(nomeBytes);
64
65          byte[] diretorBytes = diretor.getBytes("UTF-8");
66          Out.writeShort(diretorBytes.length);
67          Out.write(diretorBytes);
68
69          byte[] paisBytes = pais.getBytes("UTF-8");
70          Out.writeShort(paisBytes.length);
71          Out.write(paisBytes);
72
73          // Escreve a data de adi   o
```

```
74          Out.writeByte(ano_adi.getMonthValue());
75          Out.writeByte(ano_adi.getDayOfMonth());
76          Out.writeShort(ano_adi.getYear());
77
78          // Escreve o ano de lan amento
79          Out.writeShort(ano_lan.getValue());
80
81          byte[] classificacaoBytes = classificacao.getBytes("UTF
                -8");
82          Out.writeShort(classificacaoBytes.length);
83          Out.write(classificacaoBytes);
84
85          byte[] duracaoBytes = duracao.getBytes("UTF-8");
86          Out.writeShort(duracaoBytes.length);
87          Out.write(duracaoBytes);
88
89          byte[] GeneroBytes = Genero.getBytes("UTF-8");
90          Out.writeShort(GeneroBytes.length);
91          Out.write(GeneroBytes);
92      }
93
94      @Override
95      public void readExternal(ObjectInput dataIn) throws
            IOException {
96          try {
97              // L   os dados do fluxo de entrada
98              lapide = dataIn.readBoolean();
99              id = dataIn.readInt();
100
101             byte[] tipoBytes = new byte[dataIn.readShort()];
102             dataIn.readFully(tipoBytes);
103             tipo = new String(tipoBytes, "UTF-8");
104
105             byte[] nomeBytes = new byte[dataIn.readShort()];
106             dataIn.readFully(nomeBytes);
107             nome = new String(nomeBytes, "UTF-8");
108
109             byte[] diretorBytes = new byte[dataIn.readShort()];
110             dataIn.readFully(diretorBytes);
111             diretor = new String(diretorBytes, "UTF-8");
112
113             byte[] paisBytes = new byte[dataIn.readShort()];
114             dataIn.readFully(paisBytes);
115             pais = new String(paisBytes, "UTF-8");
116
117             int mes = dataIn.readByte();
118             int dia = dataIn.readByte();
119             int ano = dataIn.readShort();
120             ano_adi = LocalDate.of(ano, mes, dia);
```

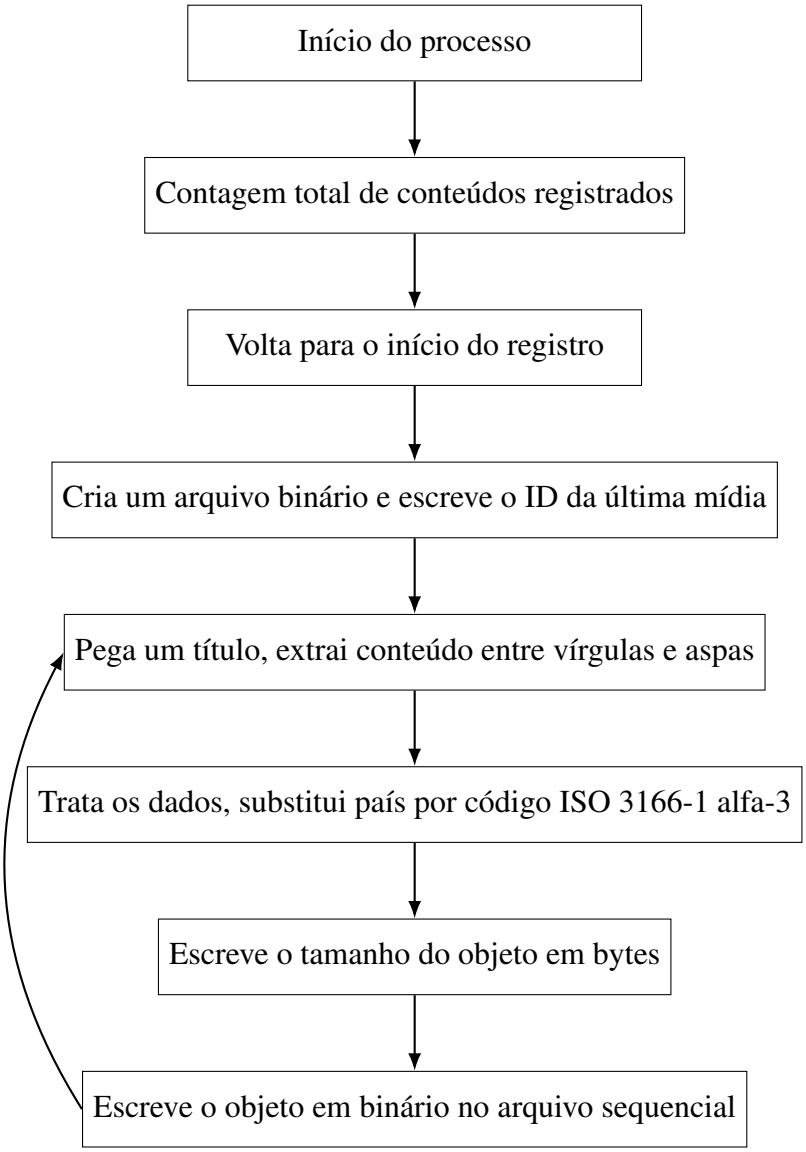
```
121
122             int anoLan = dataIn.readShort();
123             ano_lan = Year.of(anoLan);
124
125             byte[] classificacaoBytes = new byte[dataIn.
                    readShort()];
126             dataIn.readFully(classificacaoBytes);
127             classificacao = new String(classificacaoBytes, "UTF
                    -8");
128
129             byte[] duracaoBytes = new byte[dataIn.readShort()];
130             dataIn.readFully(duracaoBytes);
131             duracao = new String(duracaoBytes, "UTF-8");
132
133             byte[] GeneroBytes = new byte[dataIn.readShort()];
134             dataIn.readFully(GeneroBytes);
135             Genero = new String(GeneroBytes, "UTF-8");
136         } catch (IOException e) {
137             throw new IOException("Erro ao ler objeto Filmes", e
                    );
138         }
139     }
140
141     public String CriterioLista(int criterios){
142         // Retorna o valor do atributo correspondente ao
                crit rio fornecido
143         switch(criterios){
144             case 1: return tipo;
145             case 3: return diretor;
146             case 4: return pais;
147             case 5:{
148                 LocalDate dataTmp = ano_adi;
149                 // Formata a data de  adi   o para o formato "M/
                        d/yyyy"
150                 DateTimeFormatter format = DateTimeFormatter.
                        ofPattern("M/d/yyyy");
151                 return dataTmp.format(format);
152             }
153             case 6: return ano_lan.toString();
154             case 7: return classificacao;
155             case 9: return Genero;
156             default: return null;
157         }
158     }
159
160     @Override
161     public int compareTo(Filmes f) {
162         // Compara os filmes pelo ID
163         return Integer.compare(this.id, f.id);
```

```
164     }
165
166     public void Ler(){
167         // Exibe os dados do filme
168         System.out.println("
                ----------------------------------------");
169         System.out.println("ID: " + id);
170         System.out.println("Nome: " + nome.trim());
171         System.out.println("Ano de Lancamento: " + ano_lan);
172         System.out.println("Data de Adicao: " + ano_adi);
173         System.out.println("D u r a   o: " + duracao.trim());
174         System.out.println("Diretor: " + diretor.trim());
175         System.out.println("Pais: " + pais.trim());
176         System.out.println("G n e r o: " + Genero.trim());
177         System.out.println("Tipo: " + tipo.trim());
178         System.out.println("Faixa Etaria: " + classificacao.trim
                ());
179         System.out.println("
                ----------------------------------------");
180     }
181 }
```

2.2.1. Leitura e Escrita

Para efetuar a leitura dos dados presentes no arquivo e transformá-los em um arquivo sequencial, efetuamos o seguintes processos nesta ordem:

Fluxo de Processamento de Conteúdo



Codigo JAVA

```
1
2  // Contagem dos registros do arquivo de filmes
3         try (BufferedReader contagem = new BufferedReader(new
              FileReader(file))) {
4              contagem.readLine(); // Pula o cabe alho
5              while (contagem.readLine() != null) {
6                  contador++;
7              }
```

```
8           } catch (IOException e) {
9               System.out.println("Erro ao contar registros: " + e.
                    getMessage());
10              return;
11          }
12
13          // Processamento do arquivo de filmes
14          try (
15              BufferedReader leitura = new BufferedReader(new
                    FileReader(file));
16              RandomAccessFile out = new RandomAccessFile(
                    binarioFile, "rw");
17          ) {
18
19              File fileExistente = new File(binarioFile);
20              if (fileExistente.exists()) {
21                  fileExistente.delete(); // Exclui o arquivo
                        existente
22              }
23
24              leitura.readLine(); // Pula o cabe alho
25              out.writeInt(contador); // Escreve o n mero total
                    de registros no in cio do arquivo
26
27              String line;
28              int registro = 0;
29
30              while ((line = leitura.readLine()) != null) {
31                  // Limpeza da linha para remover caracteres
                        indesejados
32                  line = line.replaceAll(";", "").trim();
33                  line = line.replaceAll("^\"|\"$", "").trim();
34                  line = line.replaceAll("\"\"", "\"").trim();
35
36                  List<String> dadosFilme = extrairDadosLinha(line
                        );
37
38                  // Substitui o nome do pa s pelo c digo
                        correspondente
39                  String paisFilme = dadosFilme.get(4);
40                  dadosFilme.set(4, Pesquisar.PesquisarPaisAbre(
                        binarioPais, paisFilme));
41                  registro++;
42
43                  // C r i a   o do objeto Filme
44                  Filmes tmp = new Filmes(dadosFilme, registro,
                        false);
45
46                  // Serializa o objeto Filme em bytes
```

```
47                  ByteArrayOutputStream baos = new
                        ByteArrayOutputStream();
48                  try (ObjectOutputStream oos = new
                        ObjectOutputStream(baos)) {
49                      oos.writeObject(tmp);
50                  }
51                  byte[] tmpSize = baos.toByteArray();
52
53                  long posicaoAtual = out.getFilePointer();
54
55                  // Escreve o tamanho do objeto serializado
56                  out.writeInt(tmpSize.length);
57
58                  // Escreve os bytes do objeto no arquivo
                        bin rio
59                  out.write(tmpSize);
60
61              }
```

2.2.2. CRUD

O CRUD e uma parte essencial de nosso trabalho sendo uma das maiores, nele o usuário pode pesquisar, inserir novos dados, editar dados existentes e apagar aqueles que não os interessa.Sendo implementada na forma de menus na qual o usuário escolhe o que fazer.

Pesquisa

A Pesquisa e feita através do ID, que ao ser obtido e feito a pesquisa no arquivo binário e ao ser encontrado e mostra na tela

Remoção

Muito semelhante a pesquisa a remoção e feita de uma maneira bem simples, e feita uma pesquisa e que o conteúdo ser encontrado e salvo em uma variável temporária, e adicionado como ”TRUE” na variável lapide, e por fim atualizado no arquivo binário, caso o ultimo Filme/Serie for excluído e atualizado o numero que mostra o ultimo ID

Atualização

Ao usuário selecionar a atualização ele realiza uma pesquisa, e logo em seguida um sub-menu e apresentado a ele mostrando todas as opções que ele pode editar, sendo:

- Nome
- Nome do Diretor
- Data de Adição ao Catalogo
- Ano de Lançamento
- Duração, sendo minutos para filmes e temporadas para series
- Gêneros
- Classificação

- Tipo, se e um filme ou serie

Apos a atualização, a função remoção e chamada para o antigo objeto que e apagado e o novo e inserido, atualizando também o numero que representa o ultimo objeto inserido.

Inserção

A inserção e feita de um método semelhante, apos o utilizador escolher essa opção e pedido com que insira os dados nesta ordem:

- Nome
- Nome do Diretor
- Data de Adição ao Catalogo
- Ano de Lançamento
- Duração, sendo minutos para filmes e temporadas para series
- Gêneros
- Classificação
- Tipo, se e um filme ou serie

Como conclusão, o ID e obtido conseguindo através da soma do maior ID presente no arquivo com 1, e e mostrado na tela o resultado, alem disso e perguntado a pessoa se esta tudo correto caso esteja o objeto e adicionado ao arquivo binário e atualiza ID do ultimo objeto no arquivo, caso contrario a função atualização e chamada.

2.2.3. Resultado

Como resultado do arquivo sequencial e comum o arquivo binário ser menor do que o arquivo CSV original, no nosso trabalho o tamanho reduziu de 1304 KB para 1124 KB, uma redução de 24.81%.Sendo CRUD outro fator de extrema importancia devido ao fato de que, a grande maioria dos arquivo tem muandnça, sejam elas a insercao de um novo dado, a remocao, a alteração e ate mesmo a leitura de um dado.

2.3. TP2

O TP2 tem como proposta a implementação de arquivos de indexação como arvore (Arvore B, B+ ou B*), índices (Hashing Estendido) e duas lista para funcionar com Lista Invertida utilizando tudo aquilo que foi implementado no TP1, so que ao invés de se trabalhar com 2 arquivos agora se trabalha com 3, o banco de dados, o arquivo binário e o arquivo de indexação .

2.3.1. Arvore

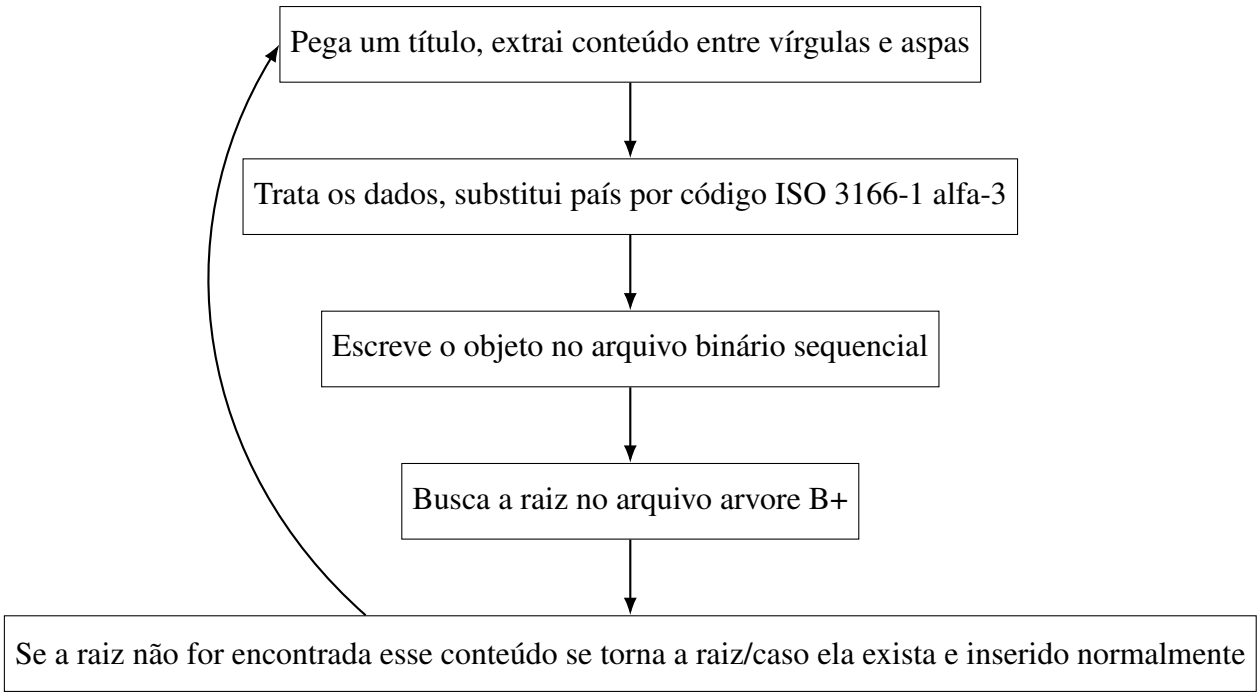
Nesta etapa, optamos pela utilização da árvore B+ devido à sua facilidade de implementação, versatilidade e alto desempenho. Esses dois últimos aspectos representam um diferencial importante, já que a árvore B+ permite que o número de filhos de cada nó seja definido pelo usuário, podendo variar de 2 até 100, o que proporciona maior flexibilidade conforme a necessidade do sistema.

Apesar de dinâmica, a implementação da árvore B+ ainda é um processo complexo que exige atenção. Em nosso projeto, estruturamos da seguinte forma: durante a leitura e escrita dos dados, no momento em que ocorre a gravação no arquivo binário, o código também escreve os mesmos dados em um arquivo destinado à árvore B+. Utilizamos os IDs como chaves e ponteiros que indicam a posição dos filhos no mesmo arquivo. Caso o nó não possua filhos, o ponteiro recebe o valor -1. Além disso, realizamos os devidos ajustes, como rotações e redistribuições, sempre que necessário, para manter a estrutura balanceada.

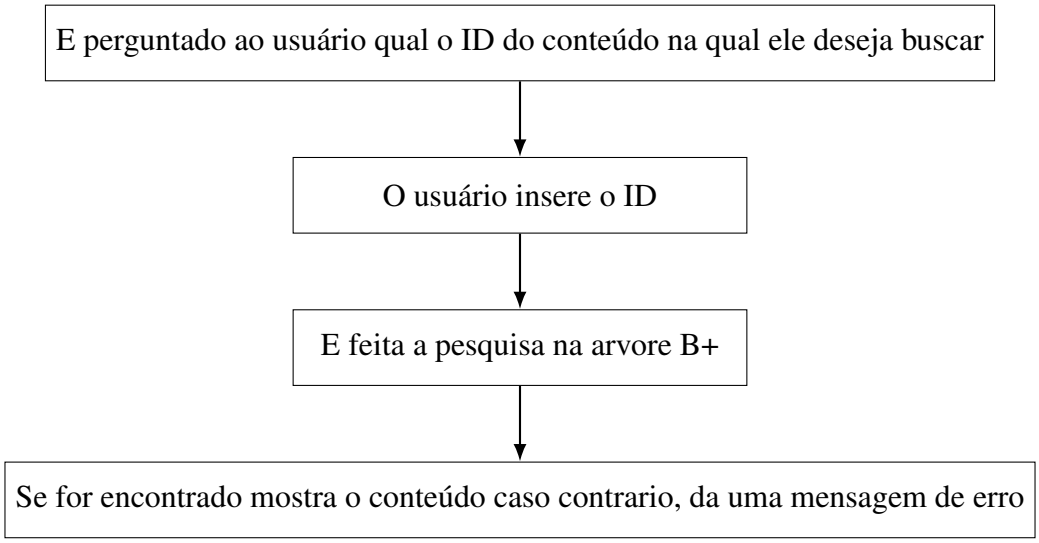
Na operação de busca, o usuário insere o ID desejado, e o código percorre a árvore até localizar o filme ou série correspondente. Caso o conteúdo não seja encontrado, é exibida uma mensagem de erro. A remoção também se baseia na busca: após localizar o conteúdo, ele é excluído do arquivo e os ponteiros da árvore são atualizados adequadamente.

Por fim, a inserção de novos registros reutiliza grande parte do processo de escrita já desenvolvido. Assim como no TP1, o usuário insere um novo objeto, e ele é adicionado tanto ao arquivo binário quanto à árvore B+, utilizando a mesma função de escrita, com os devidos ajustes na estrutura da árvore.

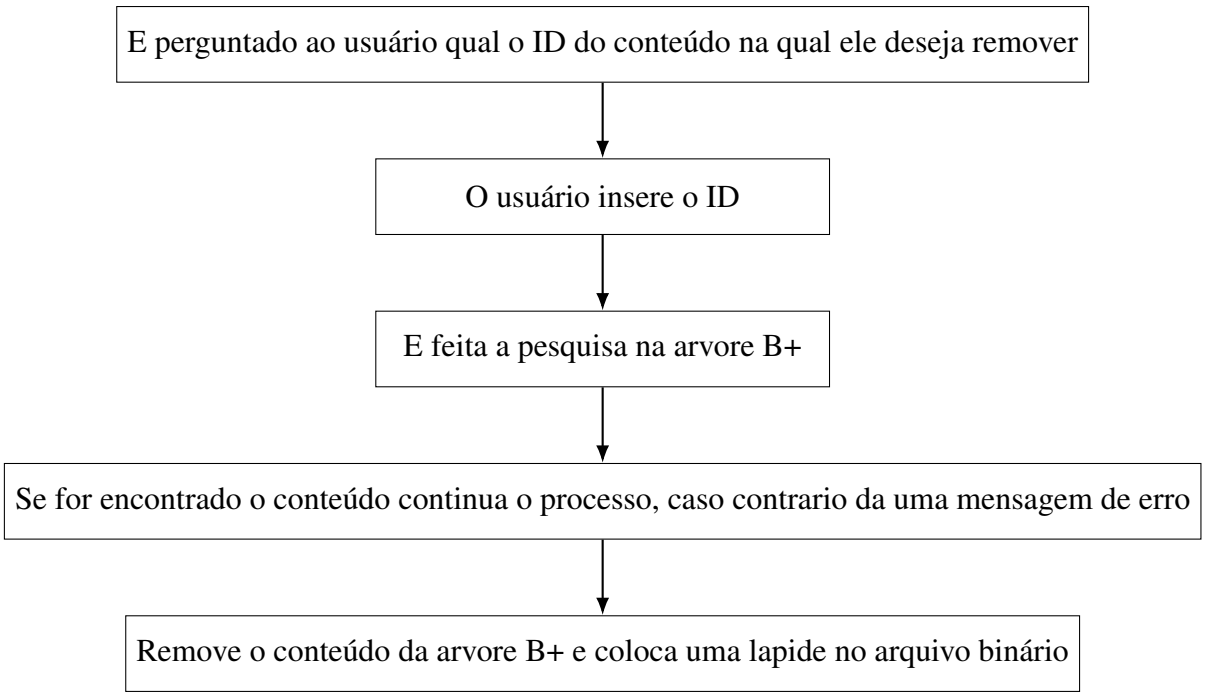
Fluxo de Processamento de Conteúdo - Leitura e Escrita



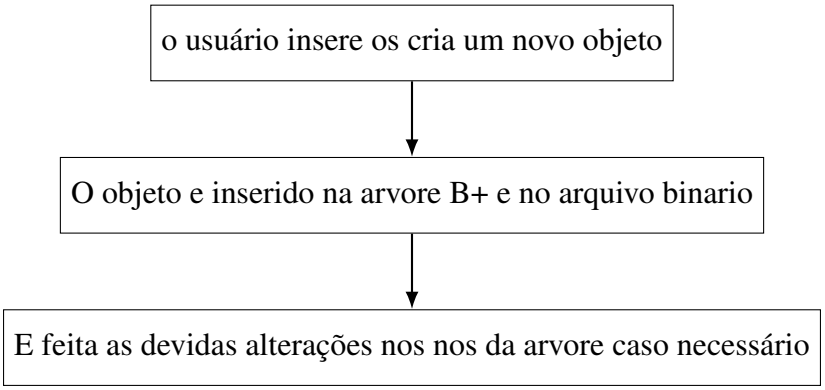
Fluxo de Processamento de Conteúdo - Pesquisa



Fluxo de Processamento de Conteúdo - Remoção



Fluxo de Processamento de Conteúdo - Adição



2.3.2. Hashing Estendido

O hash extensível se destaca por evitar colisões de forma eficiente e por permitir que a estrutura se expanda conforme a necessidade, sem desperdício significativo de espaço.

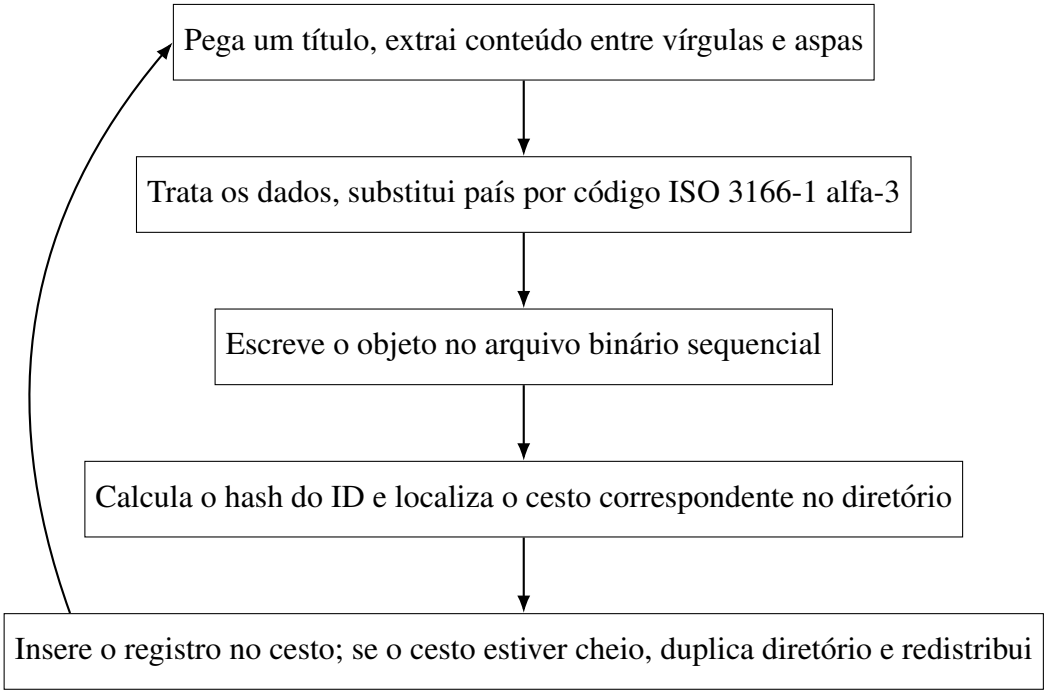
Em nosso projeto, a implementação do hash extensível foi estruturada da seguinte forma: ao realizar a gravação de um novo registro no arquivo binário, o código também insere o mesmo registro em arquivos específicos do hash, utilizando o ID como chave. O diretório do hash armazena ponteiros para os cestos, que são responsáveis por guardar os registros. Cada cesto possui uma profundidade local, e o diretório possui uma profundidade global, permitindo que a estrutura se ajuste automaticamente conforme a quantidade de dados aumenta.

Quando um cesto atinge sua capacidade máxima, o diretório pode ser duplicado e os cestos redistribuídos, garantindo que a performance das operações seja mantida mesmo com grandes volumes de dados. Durante a busca, o usuário informa o ID desejado, e o algoritmo calcula o hash para localizar rapidamente o cesto correspondente e, em seguida, o registro. Caso o registro não seja encontrado, uma mensagem de erro é exibida.

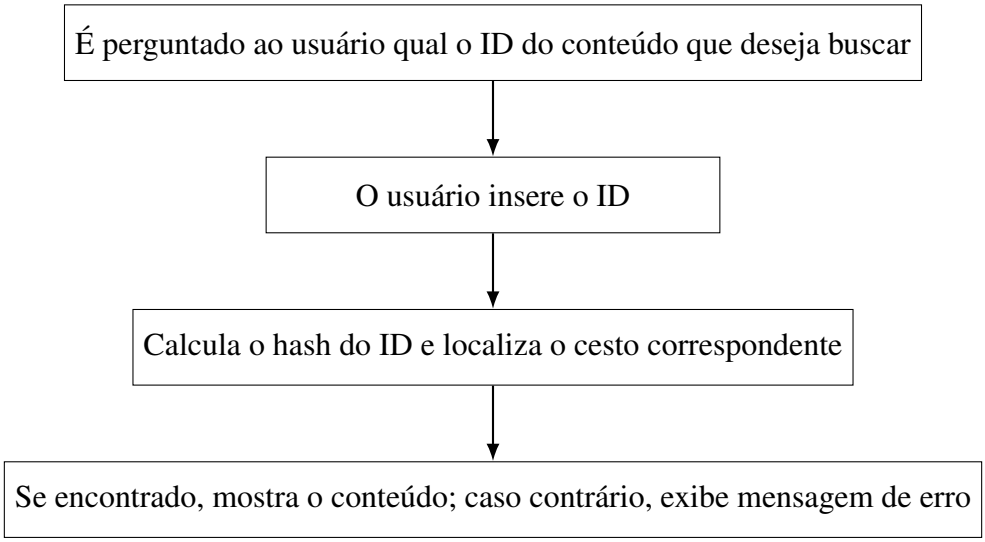
A remoção de registros segue o mesmo princípio: o sistema localiza o cesto pelo hash do ID e remove o registro, atualizando o cesto no arquivo. A inserção de novos registros reutiliza parte do processo de escrita, garantindo que tanto o arquivo binário quanto a estrutura de hash estejam sempre sincronizados.

Dessa forma, o hash extensível proporciona uma solução eficiente, flexível e escalável para a indexação e gerenciamento dos registros no sistema, facilitando operações rápidas e seguras mesmo com grandes volumes de dados.

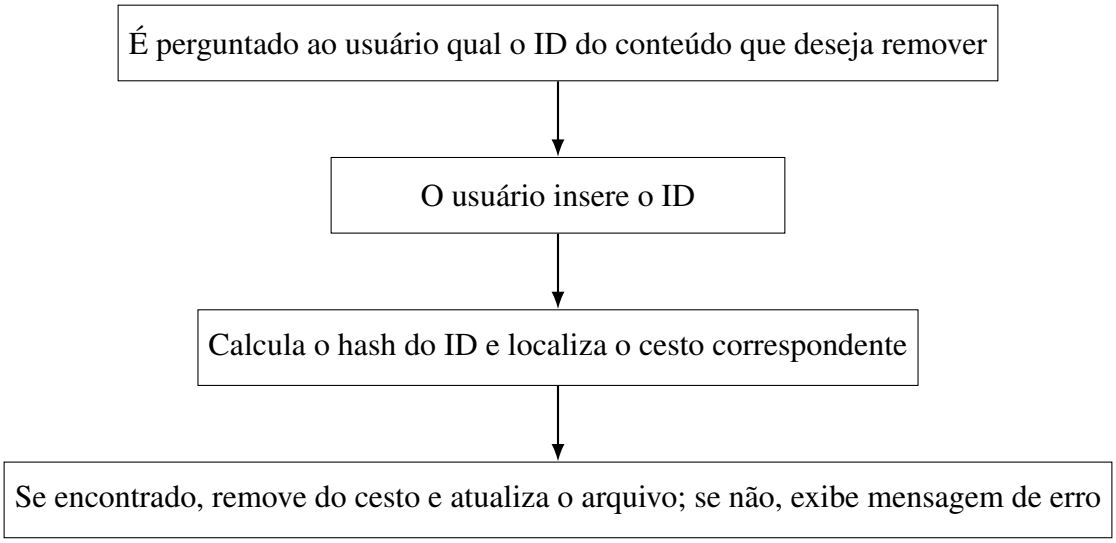
Fluxo de Processamento de Conteúdo - Leitura e Escrita (Hash Extensível)



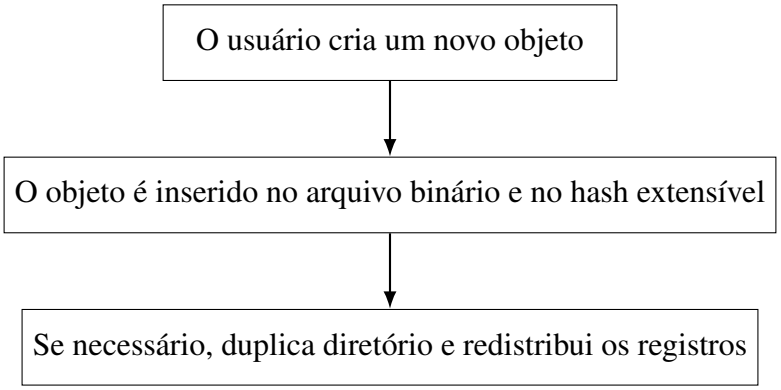
Fluxo de Processamento de Conteúdo - Pesquisa (Hash Extensível)



Fluxo de Processamento de Conteúdo - Remoção (Hash Extensível)



Fluxo de Processamento de Conteúdo - Adição (Hash Extensível)



2.3.3. Lista Invertida

A lista invertida se destaca por permitir a associação de uma chave (por exemplo, um termo, categoria ou atributo) a diversos registros, facilitando buscas rápidas e operações de filtragem em grandes volumes de dados.

Em nosso projeto, a lista invertida foi estruturada da seguinte forma: para cada chave, escolhida pelo usuário entre essas opções:

- Nome do Diretor
- Data de Adição ao Catalogo
- Ano de Lançamento
- Duração, sendo minutos para filmes e temporadas para series
- Gêneros
- Classificação

Mantemos um dicionário que armazena o nome da chave e um ponteiro para o início de uma lista de blocos. Cada bloco contém um conjunto de elementos, onde cada

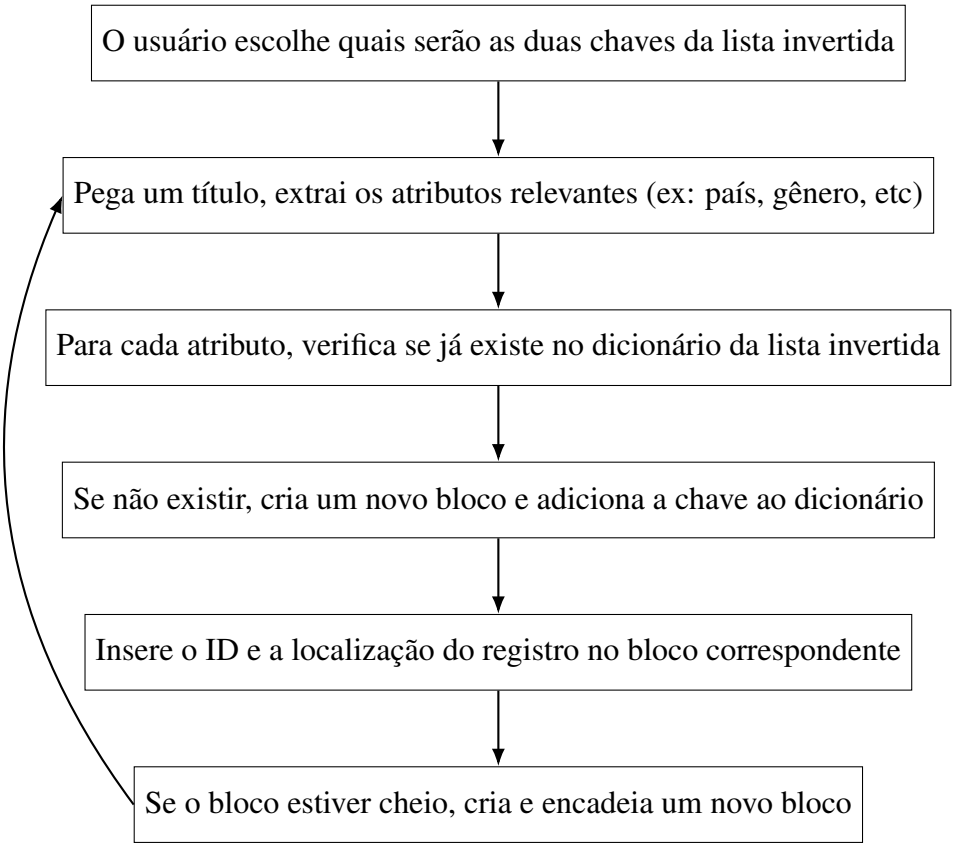
elemento representa um ID e sua localização no arquivo de dados principal. Os blocos são encadeados, permitindo que a lista cresça dinamicamente conforme a quantidade de registros associados à chave aumenta. Caso um bloco atinja sua capacidade máxima, um novo bloco é criado e encadeado ao anterior.

Durante a inserção de um novo registro, o sistema verifica se a chave já existe no dicionário. Se não existir, um novo bloco é criado e a chave é adicionada ao dicionário. O novo elemento é então inserido no bloco correspondente, mantendo a lista ordenada por ID para facilitar buscas e remoções. Se o bloco estiver cheio, um novo bloco é alocado e encadeado à lista existente.

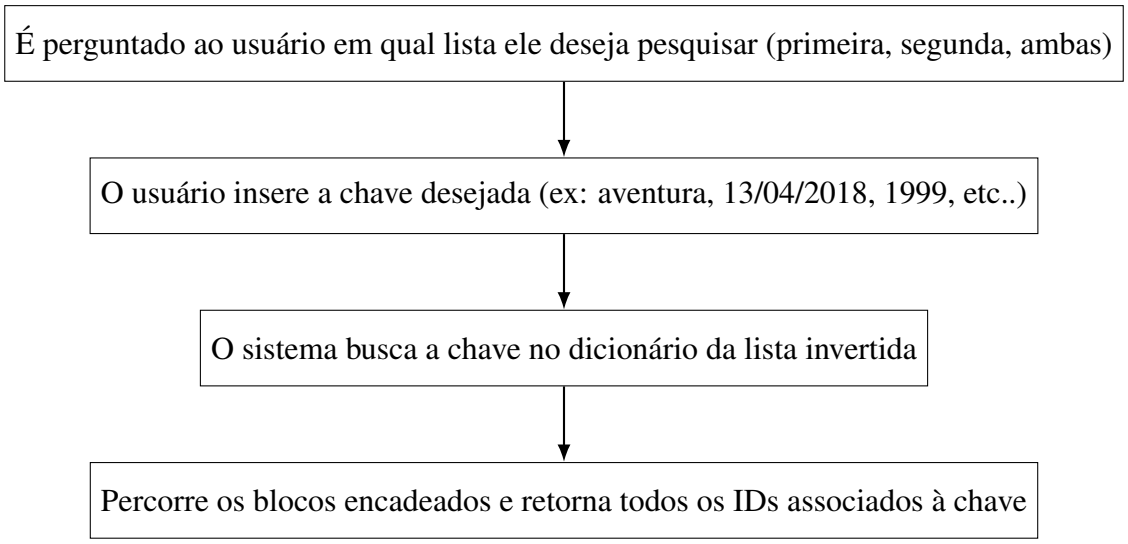
Na operação de busca, o usuário informa a chave desejada, e o sistema percorre os blocos associados a essa chave, retornando todos os IDs e localizações correspondentes. Caso a chave não seja encontrada, uma lista vazia é retornada. A remoção de registros também é eficiente: o sistema localiza o bloco correspondente e remove o elemento desejado, reorganizando a lista conforme necessário.

Por fim, a lista invertida oferece flexibilidade para operações de exclusão global (removendo um ID de todas as chaves) e limpeza total de uma chave, além de permitir a impressão de todas as listas e seus elementos para fins de auditoria ou depuração. Essa estrutura garante alto desempenho em operações de leitura e escrita, sendo especialmente útil em cenários de indexação e busca reversa.

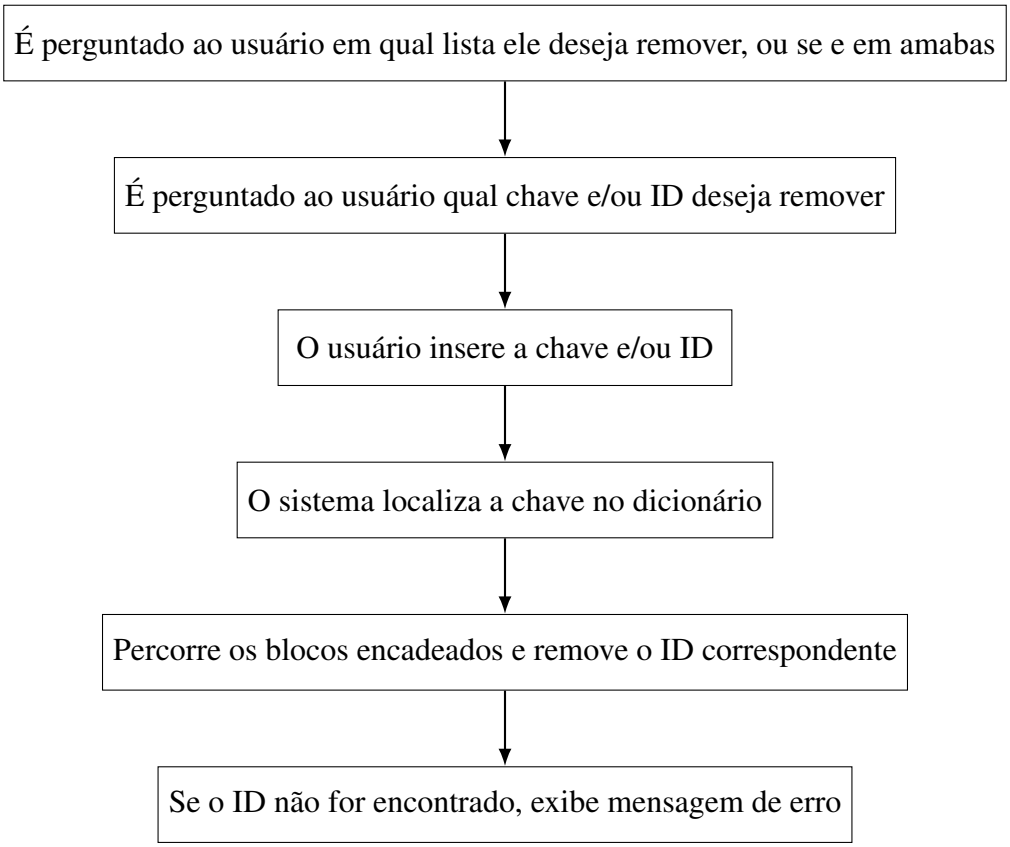
Fluxo de Processamento de Conteúdo - Leitura e Escrita (Lista Invertida)



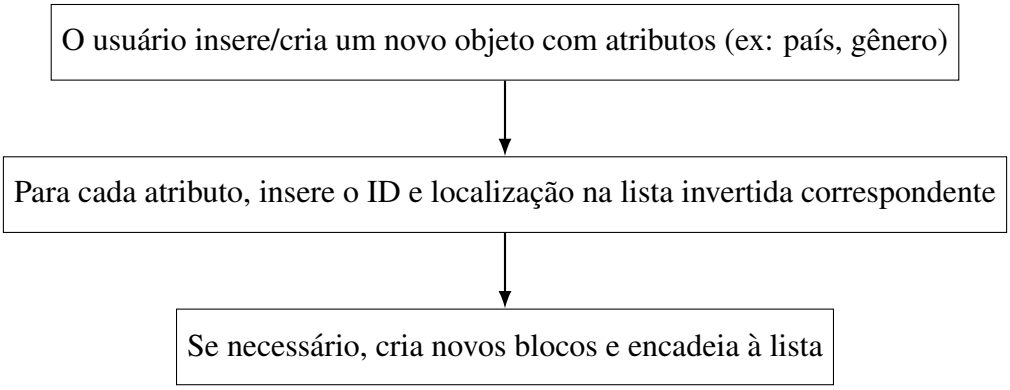
Fluxo de Processamento de Conteúdo - Pesquisa (Lista Invertida)



Fluxo de Processamento de Conteúdo - Remoção (Lista Invertida)



Fluxo de Processamento de Conteúdo - Adição (Lista Invertida)



2.3.4. Resultado

Como resultado da implementação de arquivos de indexação melhoramos em muito o tempo que se leva para buscar conteúdos em uma maneira eficiente, que comparado a anterior gasta muito menos recurso computacional.

2.4. TP3

Esta parte tem como proposta a implementação de compactação e descompactação de dados, utilizando Huffman e LZW, além de dois métodos de casamento de padrões, na parte já existente do código.

2.4.1. Compactação e Descompactação

Huffman

O principal diferencial do Huffman é sua capacidade de reduzir o tamanho dos arquivos ao atribuir códigos menores para os bytes mais frequentes, proporcionando uma compressão eficiente e sem perdas, o que é fundamental para otimizar o armazenamento e a transmissão de informações no sistema.

A implementação do algoritmo de Huffman, necessita de atenção em detalhes como a construção da árvore de códigos e o mapeamento correto entre símbolos e seus códigos binários. Em nosso projeto, estruturamos o processo da seguinte forma: durante a leitura dos dados, o sistema calcula a frequência de cada byte presente no conteúdo a ser compactado. Com essas frequências, é construída uma árvore de Huffman, onde cada nó representa um símbolo e sua frequência, e os caminhos da raiz até as folhas determinam os códigos binários de cada símbolo.

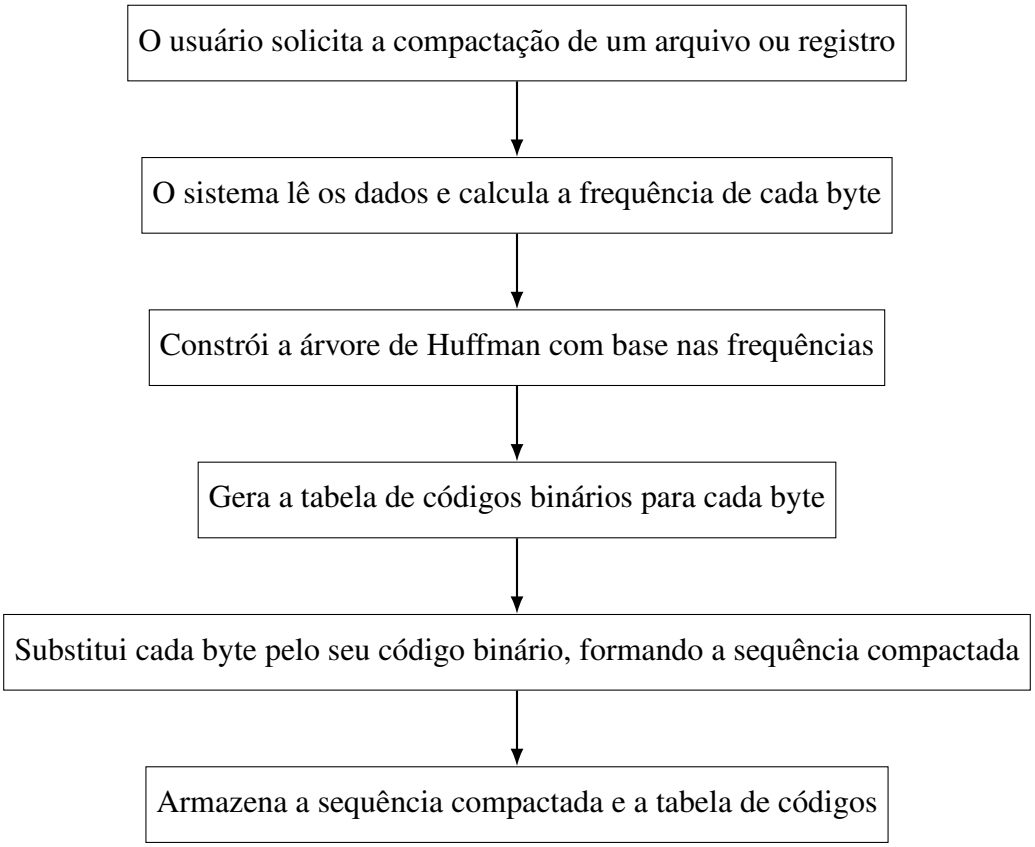
No momento da escrita, cada byte do conteúdo original é substituído pelo seu respectivo código de Huffman, gerando uma sequência compactada de bits. Essa sequência é armazenada em um vetor de bits, reduzindo significativamente o espaço ocupado em disco. Para garantir a correta descompactação, a tabela de códigos gerada é mantida e utilizada posteriormente no processo inverso.

Na operação de descompactação, o sistema utiliza a tabela de códigos para reconstruir o conteúdo original a partir da sequência de bits compactada. O algoritmo percorre

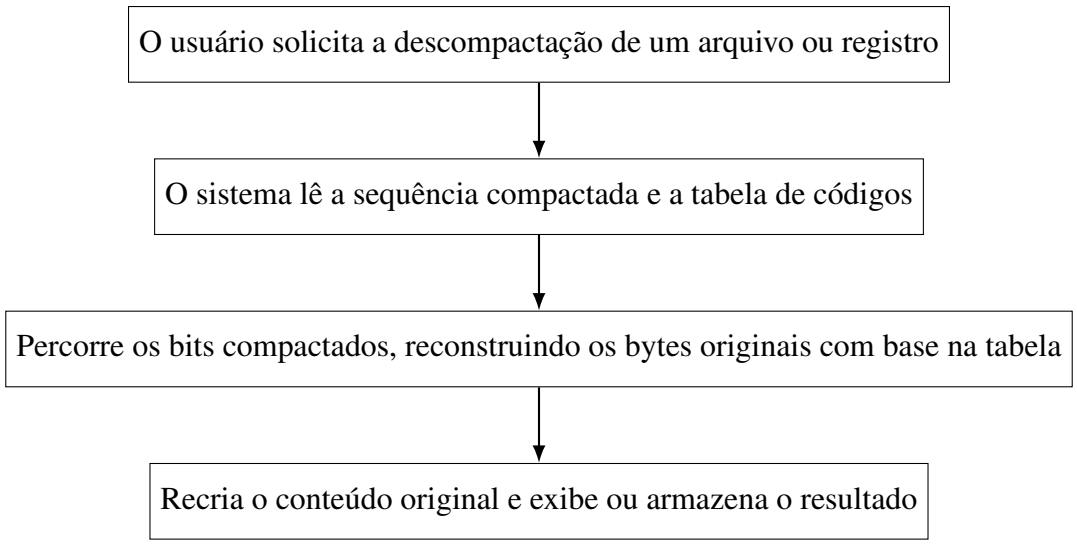
os bits, identificando os padrões correspondentes a cada símbolo, até restaurar completamente os dados originais. Caso haja qualquer inconsistência ou erro na sequência, uma mensagem de erro é exibida ao usuário.

Por fim, a integração do algoritmo de Huffman ao sistema foi realizada de forma transparente, permitindo que a compactação e descompactação dos dados ocorra automaticamente durante as operações de leitura e escrita, sem a necessidade de intervenção manual do usuário. Isso garante maior eficiência e praticidade no gerenciamento dos arquivos do projeto.

Fluxo de Processamento de Conteúdo - Compressão (Huffman)



Fluxo de Processamento de Conteúdo - Descompressão (Huffman)



LZW

O LZW se destaca por não depender de tabelas de frequência prévias e por construir dinamicamente um dicionário durante o processo de compressão, o que proporciona flexibilidade e bons resultados para diferentes tipos de dados.

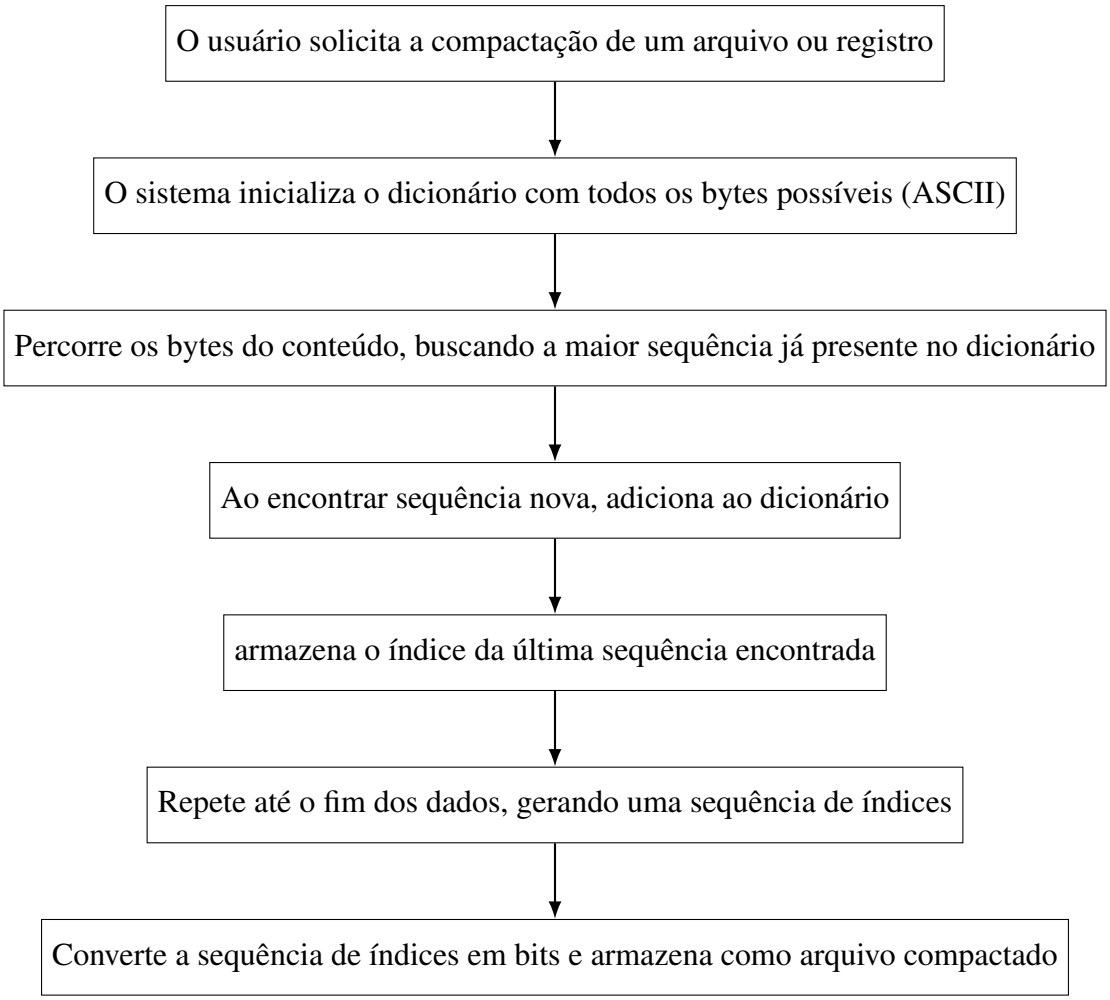
A implementação do LZW, exige atenção ao gerenciamento do dicionário e à manipulação dos índices de códigos. Em nosso projeto, estruturamos o processo da seguinte forma: durante a leitura dos dados, o sistema inicializa um dicionário contendo todas as possíveis sequências de um único byte. Em seguida, percorre o conteúdo a ser compactado, buscando sempre a maior sequência de bytes já presente no dicionário. Quando uma nova sequência é encontrada, ela é adicionada ao dicionário e o índice correspondente à última sequência conhecida é armazenado como parte da saída compactada.

No momento da escrita, os índices gerados são convertidos em uma sequência de bits, que é então armazenada em um vetor de bytes, reduzindo significativamente o espaço ocupado em disco. O tamanho dos índices é definido por uma constante, permitindo ajustar a quantidade máxima de entradas no dicionário conforme a necessidade do sistema.

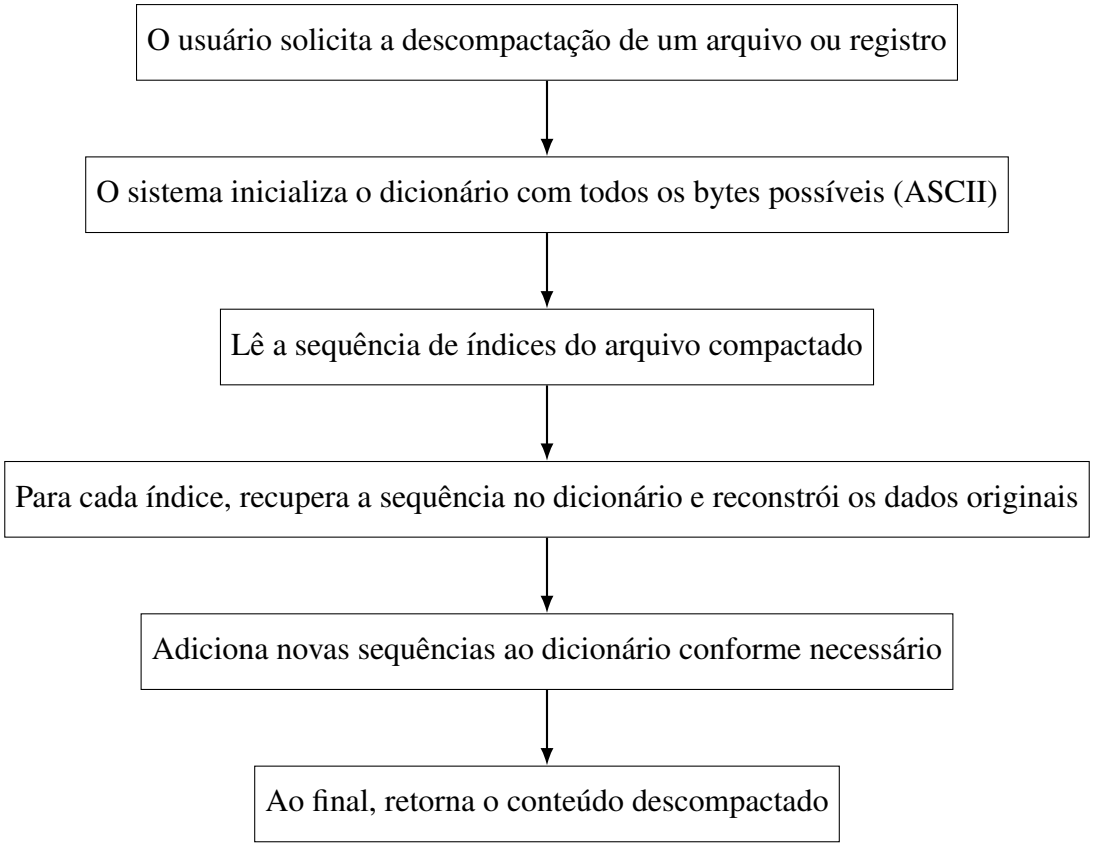
Na operação de descompactação, o sistema reconstrói o dicionário de forma idêntica ao processo de compressão, utilizando os índices para recuperar as sequências originais de bytes. O algoritmo garante que, a cada novo índice lido, a sequência correspondente seja corretamente reconstruída e adicionada ao dicionário, restaurando fielmente o conteúdo original. Caso haja qualquer inconsistência ou erro na sequência de índices, uma mensagem de erro é exibida ao usuário.

Por fim, a integração do algoritmo LZW ao sistema foi realizada de forma transparente, permitindo que a compactação e descompactação dos dados ocorra automaticamente durante as operações de leitura e escrita, sem a necessidade de intervenção manual do usuário. Isso garante maior eficiência e praticidade no gerenciamento dos arquivos do projeto.

Fluxo de Processamento de Conteúdo - Compressão (LZW)



Fluxo de Processamento de Conteúdo - Descompressão (LZW)



2.4.2. Casamento de Padrões

Na implementações de padrões nos decidimos por optar pele escolha de Boyer Moore e KMP, devido ao foto de serem as mais simples além de ser aquelas na qual temos o maior conhecimento prévio sobre como funciona e como implementar.

Boyer Moore

O principal diferencial do algoritmo de Boyer-Moore está em sua eficiência para buscas de padrões em grandes volumes de texto, utilizando heurísticas que permitem "saltos" inteligentes durante a varredura, reduzindo drasticamente o número de comparações necessárias. Isso é fundamental para acelerar pesquisas em arquivos extensos, como os utilizados no sistema.

A implementação do algoritmo de Boyer-Moore, embora baseada em conceitos clássicos, demanda atenção especial ao pré-processamento do padrão de busca. Em nosso projeto, esse processo consiste em calcular a última ocorrência de cada caractere presente no padrão, criando uma tabela que será utilizada para determinar o quanto o padrão pode ser deslocado no texto após uma falha de correspondência.

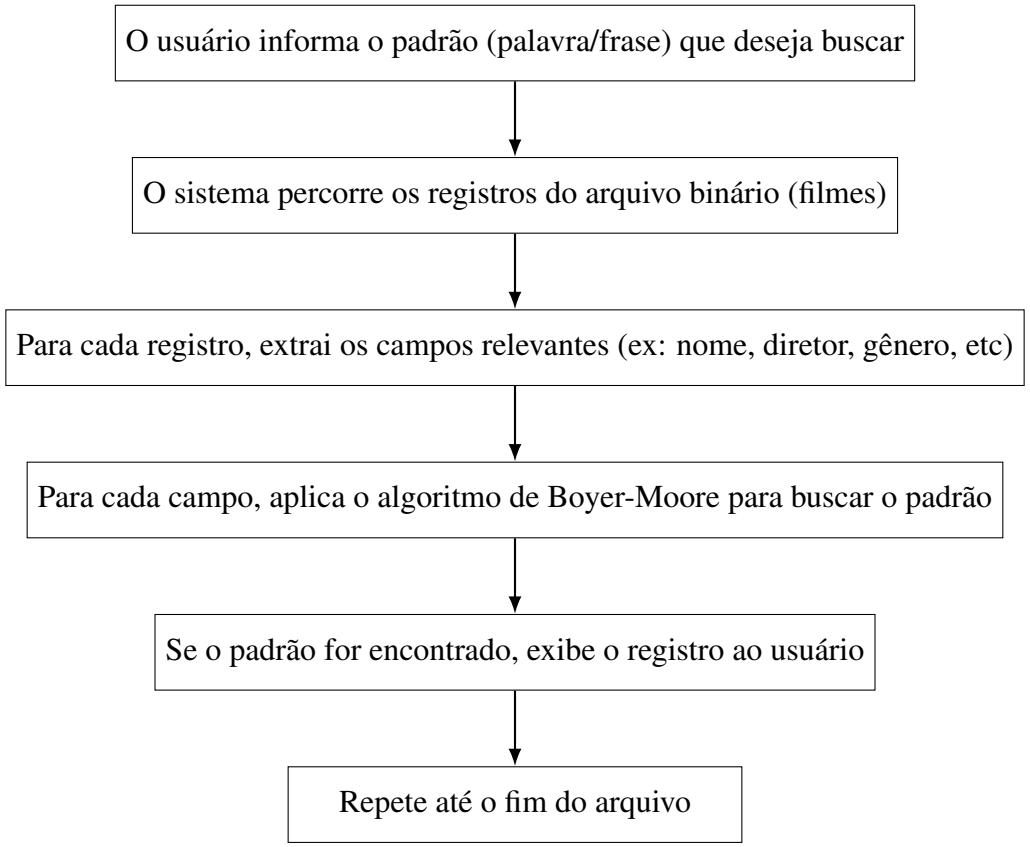
Durante a busca, o algoritmo compara o padrão com o texto da direita para a esquerda. Quando ocorre um descompasso entre os caracteres, a heurística do "caractere ruim" é aplicada: o padrão é deslocado de acordo com a posição da última ocorrência

do caractere problemático, evitando comparações desnecessárias e tornando o processo muito mais rápido do que abordagens ingênuas.

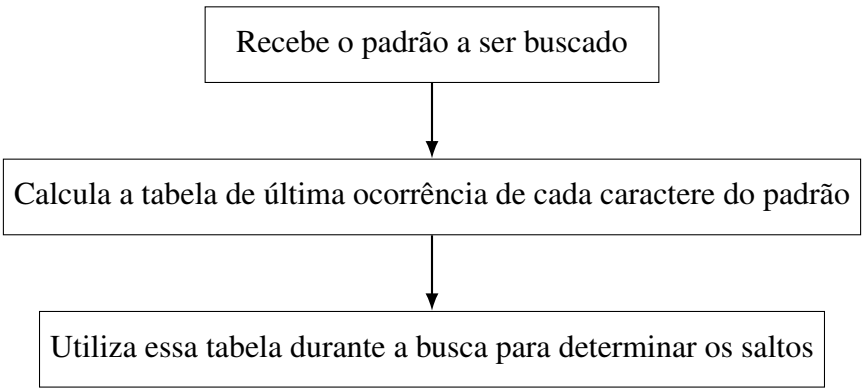
No sistema, a busca por padrões é realizada diretamente sobre os registros dos filmes armazenados em arquivos binários. Para cada filme, os campos relevantes são extraídos e convertidos em texto, e o algoritmo de Boyer-Moore é aplicado para verificar a presença do padrão desejado. Caso o padrão seja encontrado, o registro correspondente é exibido ao usuário.

A integração do Boyer-Moore ao projeto foi feita de forma modular, permitindo que buscas eficientes sejam realizadas em diferentes campos dos registros, sem a necessidade de varrer todo o conteúdo byte a byte. Isso garante maior agilidade nas operações de pesquisa e contribui para uma experiência de uso mais fluida e responsiva no sistema.

Fluxo de Processamento de Conteúdo - Pesquisa (Boyer-Moore)



Fluxo de Processamento de Conteúdo - Pré-processamento (Boyer-Moore)



KMP

A grande vantagem do algoritmo KMP (Knuth-Morris-Pratt) é sua eficiência na busca de padrões em textos, evitando retrocessos desnecessários e tornando o processo de pesquisa muito mais rápido, especialmente em grandes volumes de dados. Isso é fundamental para otimizar buscas por informações específicas dentro dos registros do sistema.

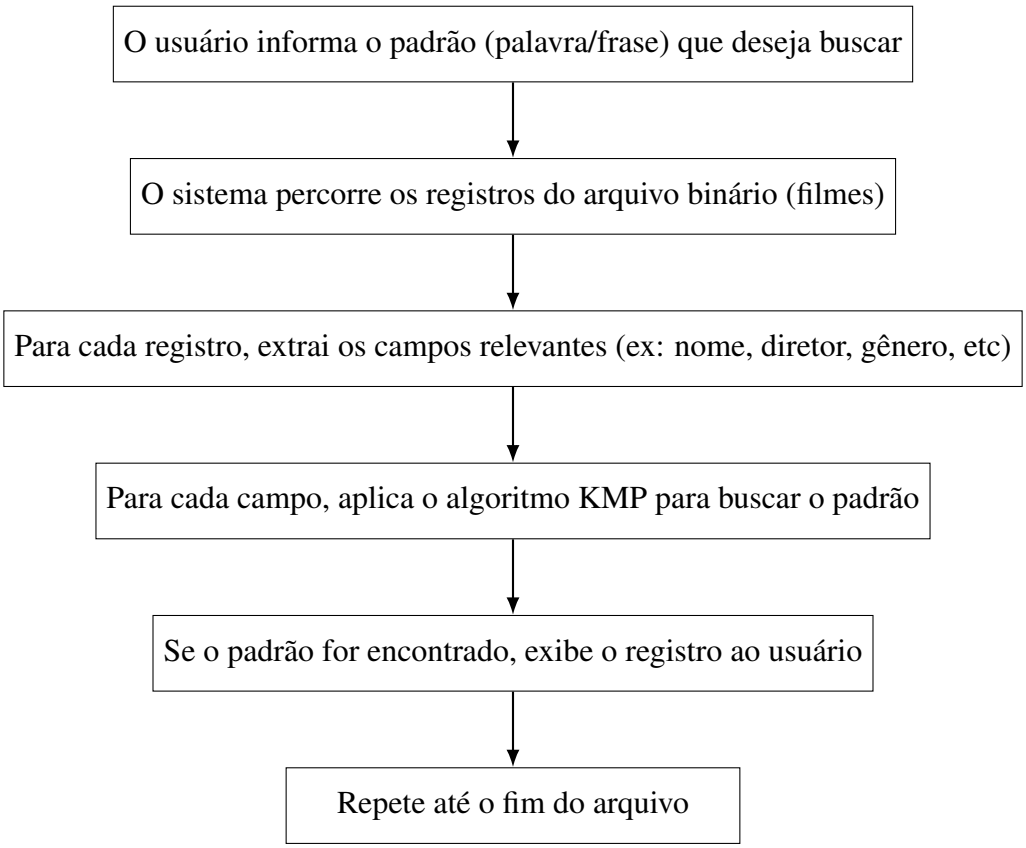
A implementação do algoritmo KMP, embora baseada em um conceito simples, exige atenção ao pré-processamento do padrão de busca. Em nosso projeto, esse processo consiste na construção do vetor LPS (Longest Prefix Suffix), que armazena o tamanho do maior prefixo do padrão que também é sufixo. Esse vetor permite que, ao ocorrer uma falha na correspondência durante a busca, o algoritmo saiba exatamente para onde deve voltar no padrão, evitando comparações repetidas.

Durante a busca, o algoritmo compara o padrão com o texto de forma sequencial. Quando ocorre uma diferença entre os caracteres, o vetor LPS é utilizado para determinar o próximo ponto de comparação, sem a necessidade de reiniciar a busca do início do padrão. Isso garante que cada posição do texto seja analisada no máximo uma vez, proporcionando uma busca eficiente.

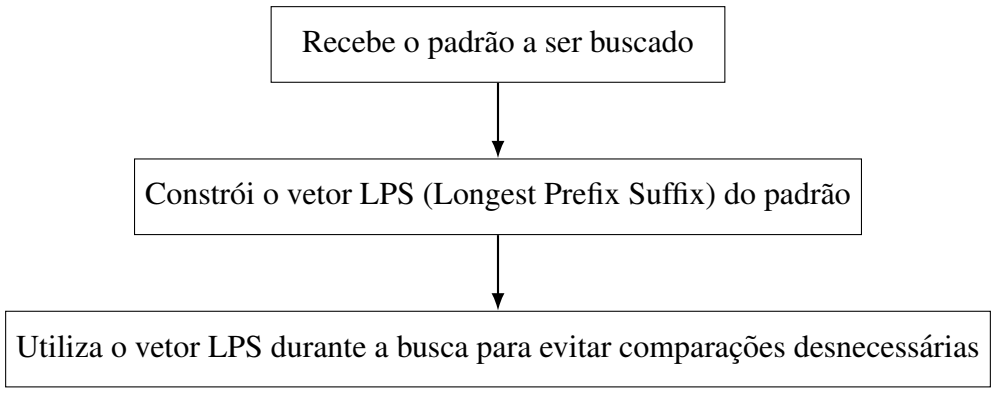
No sistema, a busca por padrões é realizada diretamente nos registros dos filmes armazenados em arquivos binários. Para cada filme, os campos relevantes são extraídos e convertidos em texto, e o algoritmo KMP é aplicado para verificar a presença do padrão desejado. Caso o padrão seja encontrado, o registro correspondente é exibido ao usuário.

A integração do KMP ao projeto foi feita de forma transparente, permitindo buscas rápidas e precisas em diferentes campos dos registros, sem a necessidade de varrer todo o conteúdo de forma ineficiente. Isso garante maior agilidade nas operações de pesquisa e contribui para uma experiência de uso mais eficiente e satisfatória no sistema.

Fluxo de Processamento de Conteúdo - Pesquisa (KMP)



Fluxo de Processamento de Conteúdo - Pré-processamento (KMP)



2.4.3. Resultados

Com isso se conclui que a implementação de compressão e um fator importante quando se trata de grandes dados visto que em nosso código, que por mais que seja pequeno teve em media uma redução de 30%. Com isso a busca de padrões e uma boa implementação, pois consegue identificar com precisão dados que tenham semelhanças com palavras nas quias você tenha interseção

2.5. TP4

Nesta ultima etapa teremos que implementar dois tipos de criptografia, uma Simples, podendo ser Substituição, Vigenére ou Transposição, e uma Moderna, sendo DES ou RSA. Nos Decidimos por optar pela de Vigenére e pela DES, devido ao fato serem a que a que mais compreendemos e conseguimos resolver facilmente tanto a criptografia quanto a descriptografia.

2.5.1. Vigenére

O principal diferencial do algoritmo de Vigenère é sua capacidade de proteger os dados por meio de criptografia simétrica, utilizando uma chave fornecida pelo usuário para embaralhar os bytes do arquivo. Isso garante confidencialidade e segurança das informações armazenadas e transmitidas no sistema, dificultando o acesso não autorizado.

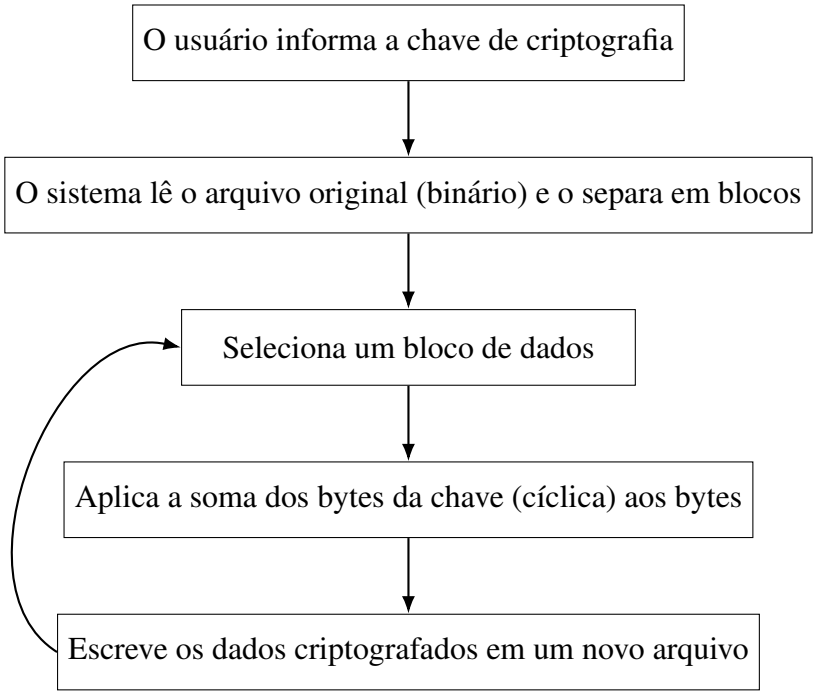
A implementação do algoritmo de Vigenère, apesar de conceitualmente simples, exige atenção ao correto tratamento dos bytes e à aplicação da chave ao longo de todo o conteúdo. Em nosso projeto, estruturamos o processo da seguinte forma: durante a criptografia, o sistema solicita ao usuário uma chave, que é convertida em um vetor de bytes. Em seguida, cada byte do arquivo original é somado ao byte correspondente da chave (de forma cíclica), gerando um novo arquivo criptografado.

No momento da gravação, tanto os metadados quanto os dados dos registros são criptografados utilizando a chave informada, garantindo que todo o conteúdo do arquivo permaneça protegido. O processo é realizado de maneira transparente, sem alterar a estrutura dos dados, apenas modificando seus valores para impedir a leitura direta.

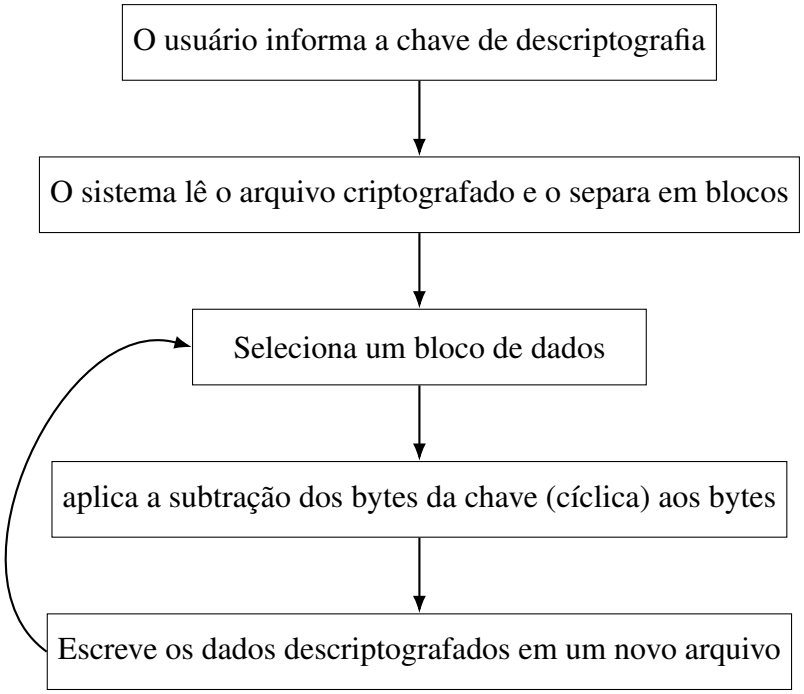
Na operação de descriptografia, o sistema solicita novamente a chave ao usuário. Utilizando o mesmo algoritmo, porém invertendo a operação, cada byte do arquivo criptografado é subtraído do byte correspondente da chave, restaurando assim o conteúdo original. Caso a chave informada esteja incorreta, os dados recuperados serão inválidos, protegendo a integridade das informações.

Por fim, a integração do algoritmo de Vigenère ao sistema foi feita de modo a permitir que a criptografia e descriptografia dos arquivos ocorram de forma simples e acessível, bastando ao usuário informar a chave desejada. Isso proporciona uma camada adicional de segurança no gerenciamento dos arquivos do projeto, sem comprometer a usabilidade ou a eficiência das operações.

Fluxo de Processamento de Conteúdo - Criptografia (Vigenère)



Fluxo de Processamento de Conteúdo - Descriptografia (Vigenère)



2.5.2. DES

O algoritimo de DES tem como seu lado forte sua capacidade de proteger os dados por meio de criptografia simétrica em blocos, utilizando uma chave secreta para embaralhar as

informações de forma robusta. Isso garante a confidencialidade dos arquivos armazenados e transmitidos no sistema, dificultando o acesso não autorizado aos dados sensíveis.

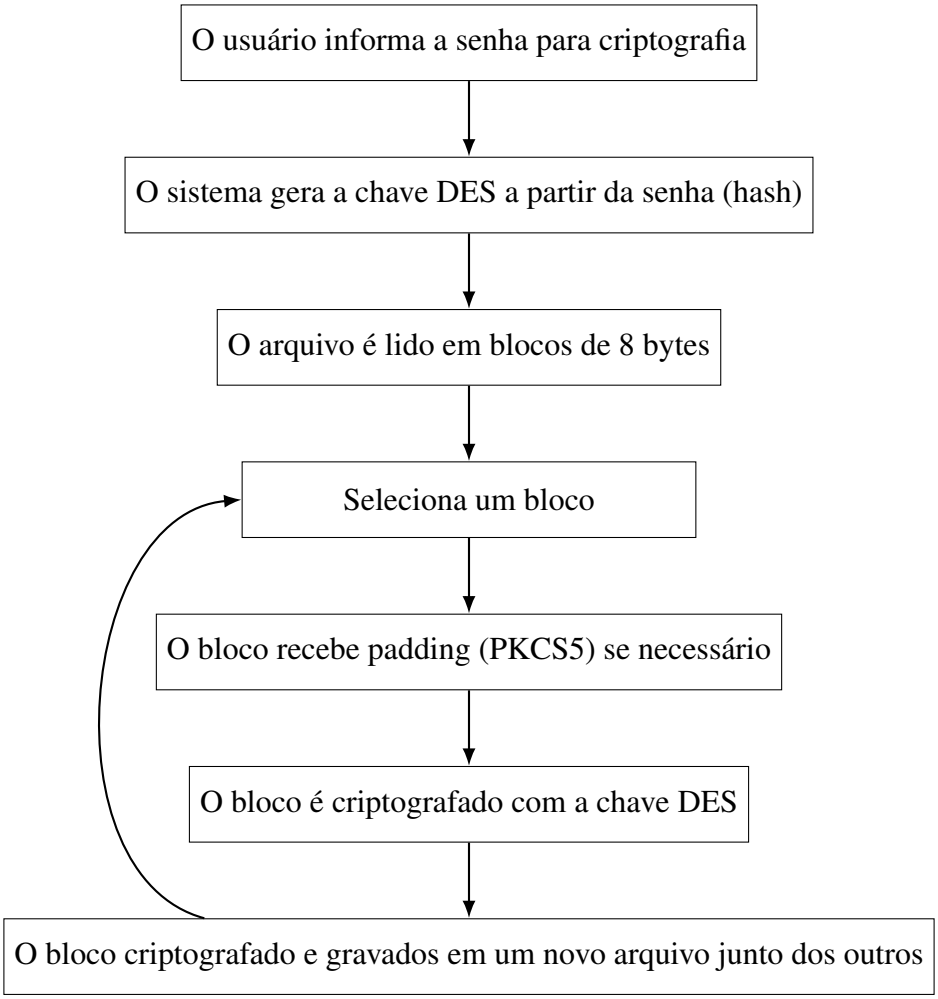
A implementação do algoritmo DES, apesar de baseada em operações matemáticas bem definidas, exige atenção a detalhes como o gerenciamento dos blocos de dados, o uso correto das permutações e substituições, além do tratamento adequado do preenchimento (padding) dos dados. Em nosso projeto, estruturamos o processo da seguinte forma: durante a criptografia, o sistema solicita ao usuário uma senha, que é convertida em uma chave de 64 bits por meio de um hash. Os dados do arquivo são então divididos em blocos de 8 bytes, e cada bloco é criptografado individualmente utilizando a chave gerada.

No momento da gravação, tanto os metadados quanto os dados dos registros são criptografados em blocos, garantindo que todo o conteúdo do arquivo permaneça protegido. O processo é realizado de maneira transparente, sem alterar a estrutura dos dados, apenas modificando seus valores para impedir a leitura direta.

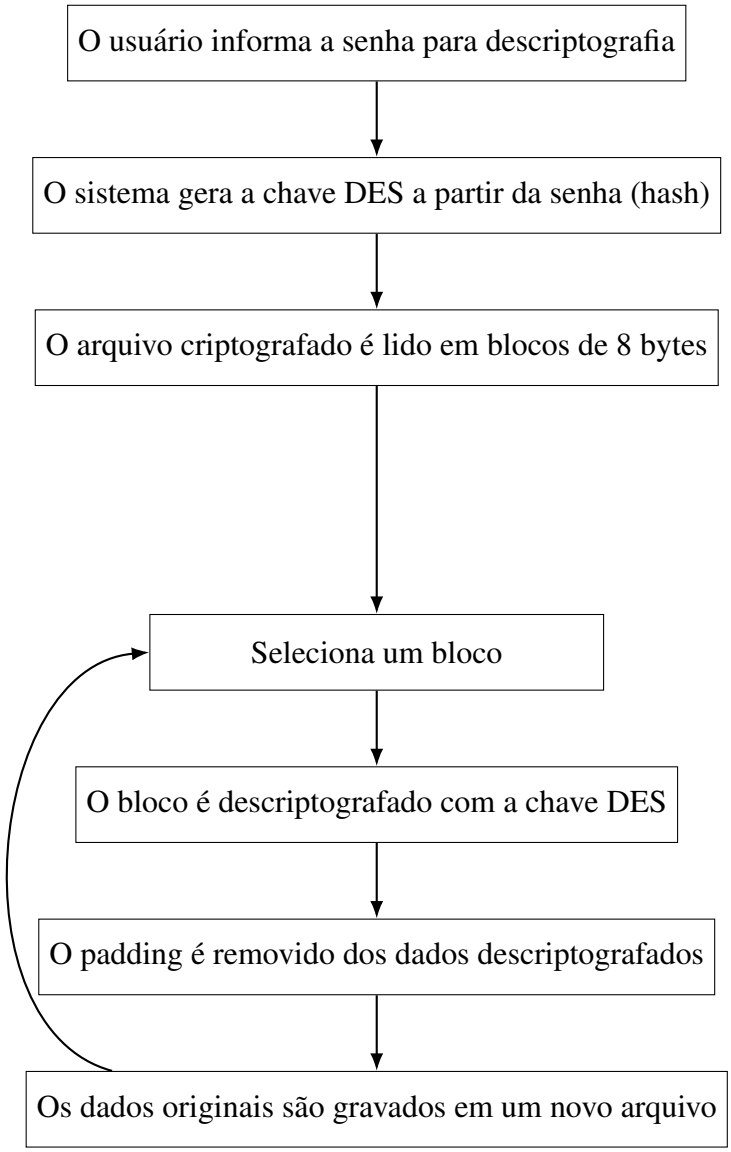
Na operação de descriptografia, o sistema solicita novamente a senha ao usuário. Utilizando a mesma chave, cada bloco criptografado é processado pelo algoritmo DES em modo reverso, restaurando assim o conteúdo original. O sistema também remove o preenchimento extra adicionado durante a criptografia, garantindo que os dados recuperados estejam íntegros. Caso a senha informada esteja incorreta, os dados descriptografados serão inválidos, protegendo a integridade das informações.

Por fim, a integração do algoritmo DES ao sistema foi feita de modo a permitir que a criptografia e descriptografia dos arquivos ocorram de forma simples e segura, bastando ao usuário informar a senha desejada. Isso proporciona uma camada adicional de segurança no gerenciamento dos arquivos do projeto, sem comprometer a usabilidade ou a eficiência das operações.

Fluxo de Processamento de Conteúdo - Criptografia (DES)



Fluxo de Processamento de Conteúdo - Descriptografia (DES)



Resultado

Por fim se conclui que as criptografias implementadas por mais que no mundo de hoje não sejam mais seguras, devido ao métodos de descriptografia avançados e ao avanço da computação se consegue quebrá-las em instantes, elas ainda servem para um proposito, que e a compreensão básico de como um criptografia funciona e como e implementada

3. Conclusão

Em suma, conclui-se que o projeto passou por diversos desafios ao longo de seu desenvolvimento, dando ênfase principalmente para o TP 2 por seu tamanho e nível de complexidade. Entretanto, a dupla obteve êxito e conseguiu realizar os códigos e suas devidas necessidades a tempo e de forma bastante clara e completa.