UNIVERSIDADE DO EXTREMO SUL CATARINENSE - UNESC

CURSO DE CIÊNCIA DA COMPUTAÇÃO

BRUNO CASEMIRO KURZAWE

FADA-CHL: FERRAMENTA PARA ANÁLISE DE DESEMPENHO E COMPARAÇÃO DOS ALGORITMOS DE COMPACTAÇÃO SEM PERDA HUFFMAN E LZW

CRICIÚMA

2012

BRUNO CASEMIRO KURZAWE

FADA-CHL: FERRAMENTA PARA ANÁLISE DE DESEMPENHO E COMPARAÇÃO DOS ALGORITMOS DE COMPACTAÇÃO SEM PERDA HUFFMAN E LZW

Trabalho de Conclusão de Curso, apresentado para obtenção do grau de Bacharel no curso de Ciência da Computação da Universidade do Extremo Sul Catarinense, UNESC.

Orientador(a): Profª MSc. Christine Vieira Scarpato

CRICIÚMA

2012

BRUNO CASEMIRO KURZAWE

FADA-CHL: FERRAMENTA PARA ANALISE DE DESEMPENHO E COMPARAÇÃO DOS ALGORITMOS DE COMPACTAÇÃO SEM PERDA HUFFMAN E LZW

Trabalho de Conclusão de Curso aprovado pela Banca Examinadora para obtenção do Grau de Bacharel, no Curso de Ciência da Computação da Universidade do Extremo Sul Catarinense, UNESC, com Linha de Pesquisa em algoritmos de compactação.

Criciúma, 27 de Junho de 2012.

BANCA EXAMINADORA

Profa. MSc. Christine Vieira Scarpato

Prof. MSc. Paulo João Martins

Prof. MSc. Kristian Madeira

Aos meus pais que sempre me apoiaram minha Irma e minha namorada Júlia.

AGRADECIMENTOS

Agradeço a professora Christine Vieira Scarpato pela orientação neste trabalho, ao professor Kristian Madeira pela ajuda no inicio do projeto e proposta, ao professores que nesses anos foram importantes na minha formação. Agradeço aos colegas com quem dividi momentos bons e ruins, onde juntos desenvolvemos projetos uns que deram certo e outros que não deram.

Agradeço a professora Ana Claudia, coordenadora do curso de Ciência da Computação, e a secretaria Margarete, nunca esquecerei os conselhos tanto na vida acadêmica como na vida pessoal, agradeço também meus empregadores pela compreensão nos momentos onde precisei me ausentar desenvolver trabalhos e projetos universitários.

.

“Tudo posso naquele que me fortalece”

Felipenses 4:13

ABSTRACT

The area of compaction of data is one of the oldest areas of computing, in 1950 already studied methods for this. In the 80's and 90's this area has greatly increased with the development of algorithms that we use until today as Huffman , LEMPEL-ZIV & WELCH LZW, Burrows-Wheeler Transform BWT and PPM Prediction by Partial Matching. After to read about this subject, and know that current compressors such as WinZip and WinRar use these algorithms we interested to better know its mechanism, and through of techniques the performance and analysis techniques such as compaction rate, velocity of compression and use of memory , to find the best algorithm for each type of file. In this work we implemented a tool called FAIRY-CHL that implements the algorithm of HUFFMAN and the of LZW algorithm, this tool analyzes the results to the files of the Canterbury benchmark Corpus developed by Canterbury University in New Zealand. During development of the tool we had problems with two files to analyze because of the size of these files.

After execution the tool can obtain the most favorable results for the algorithm when applied to HUFFMAN table Canterbury Corpus it because we have, in general, files of text where the algorithm HUFFMAN has a rate compaction and better execution time, these higher rates occur by the use of binary trees. The execution time in HUFFMAN is much better than the times of the LZW algorithm, on average the algorithm HUFFMAN is 200 times faster than the LZW algorithm. About the utilization of memory used the two algorithms have a similar use. As a conclusion we can say that when applied to the table Canterbury HUFFMAN the algorithm is more efficient in all the parameters discussed and the tool developed serves the requirements and functionality for which it was developed. As future work we suggest the implementation of statistical analysis on the results obtained and to make that tool itself can generate these results.

Key words: Data compaction, algorithms compaction, analysis of performance, compaction rate and execution time.

RESUMO

A área de compactação de dados é uma das áreas mais antigas da computação, em 1950 já se estudavam métodos para isso. Nas décadas de 80 e 90 essa área teve um grande crescimento com o desenvolvimento de algoritmos que utilizamos até hoje como Huffman, LEMPEL-ZIV & WELCH LZW, Burrows-Wheeler Transform BWT e Prediction by Partial Matching PPM. Após ler sobre o assunto, e descobrir que compressores atuais como WinZip e WinRar utilizam esses algoritmos nos interessamos a conhecer melhor seu funcionamento, e através de técnicas de análise de desempenho, como a taxa de compactação, velocidade de compactação e uso da memória, descobrir o melhor algoritmo para cada tipo de arquivo. Neste trabalho implementamos uma ferramenta chamada FADA-CHL que implementa o algoritmo de HUFFMAN e o algoritmo de LZW, essa ferramenta analisa os resultados para os arquivos do benchmark da Canterbury Corpus desenvolvido pela universidade de CanterBury da Nova Zelandia. Durante o desenvolvimento da ferramenta obtivemos problemas com dois arquivos para analise por causa do tamanho desses arquivos.

Após a execução da ferramenta podemos obter os resultados mais favoráveis para o algoritmo de HUFFMAN quando aplicado a tabela de CanterBury Corpus por termos em geral arquivos de texto onde o algoritmo de HUFFMAN possui uma taxa de compactação e tempo de execução bem melhor, essas taxas maiores se dão pela utilização de arvores binárias. Os tempos de execução de HUFFMAN são muito melhores que os tempos do algoritmo de LZW, em média o algoritmo de HUFFMAN é 200 vezes mais rápido que o algoritmo de LZW. Quanto a utilização de memória os dois algoritmos possuem uma utilização parecida. Como uma conclusão podemos dizer que quando aplicados a tabela CanterBury o algoritmo de HUFFMAN é mais eficiente em todos os parâmetros abordados e a ferramenta desenvolvida atende os requisitos e funcionalidades para a qual foi desenvolvida. Como trabalhos futuros sugerimos a implementação de análise estatística sobre os resultados obtidos e fazer com que a própria ferramenta possa gerar esses resultados.

Palavras-chave: Compactação de dados, algoritmos de compactação, análise de desempenho, taxa de compactação e tempo de execução.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

RLE	Run-length encoding
LZW	LEMPEL-ZIV & WELCH
BWT	Burrows-Wheeler Transform
PPM	Prediction by Partial Matching

SUMÁRIO

1 INTRODUÇÃO

Com a expansão da transmissão de dados em todo o mundo, o crescimento das redes de computadores e a expansão da internet, todos os dias surgem novas técnicas de processamento de dados a fim de encontrar melhores maneiras de transmiti-los ou armazená-los. HELD (1992) e FRANÇA NETO (1998) afirmam em suas obras que ao mesmo tempo em que a capacidade de transmissão e armazenamento cresce, as informações a serem transmitidas e armazenadas crescem mais rápido ainda, desta forma a compactação de dados se faz útil e muito necessária.

Ao citarmos compactação e transmissão de dados logo consideramos que sua aplicação é muito importante em arquivos de imagem e vídeo, pois nesse tipo de arquivo encontramos uma quantidade grande de informação, tornando-os arquivos de tamanho elevado e de alto custo para sua transmissão e armazenamento.

Para isso buscamos na compactação de dados uma maneira de solucionar esse problema. Dentro da compactação de dados abordamos dois tipos, compactação com perda e compactação sem perda de informações. Neste projeto o foco será a aplicação dos algoritmos de compactação sem perda, será feita uma análise comparativa entre os dois principais algoritmos utilizados nessa área, que são o algoritmo de HUFFMAN e o algoritmo LEMPEL-ZIV & WELCH LZW que é chamado de LZW. Apresentaremos as vantagens e desvantagens de cada um em determinados tipos arquivos.

Este assunto possui aspectos que abrangem várias áreas de estudo dentro da computação e das engenharias em geral, os conceitos utilizados neste estudo são encontrados em muitos outros campos, entre eles podemos citar processamento de sinal, compactação de áudio e compactação de vídeo.

Neste contexto podemos dizer que a compactação de dados é de vital importância nos tempos atuais devido à grande massa de informação que tem sido gerada nas redes de computadores, como grandes redes corporativas e a internet.

Podemos perceber que os conhecimentos aplicados na compactação de dados são comuns em diversas áreas dentro da computação, engenharia e matemática, pois utilizam em sua estrutura cálculos matemáticos e probabilísticos.

1.1 OBJETIVO GERAL

Desenvolver um protótipo da ferramenta FADA-CHL para análise de desempenho dos algoritmos de compactação sem perda LEMPEL-ZIV & WELCH LZW e HUFFMAN.

1.2 OBJETIVOS ESPECÍFICOS

a) Diferenciar compressão de compactação.
b) Detalhar o funcionamento do algoritmo de Huffman.
c) Detalhar o funcionamento do algoritmo de LZW.
d) Demonstrar a diferença entre os algoritmos propostos em relação aos recursos computacionais utilizados, sendo que um utiliza árvores binárias e o outro dicionário de dados.
e) Desenvolver um protótipo de uma ferramenta chamado FADA-CHL que irá fazer a análise de desempenho dos algoritmos LZW e HUFFMAN, considerando taxa de compactação, velocidade de execução e uso de memória e processamento.
f) Realizar a comparação entre os resultados obtidos através da FADA-CHL.

1.3 JUSTIFICATIVA

A área de compactação de dados é uma das áreas mais antigas da computação, em 1950 já se estudavam métodos para isso. Nas décadas de 80 e 90 essa área teve um grande crescimento com o desenvolvimento de algoritmos que utilizamos até hoje como Huffman, LEMPEL-ZIV & WELCH LZW, Burrows-Wheeler Transform BWT e Prediction by Partial Matching PPM. Após ler sobre o assunto, e descobrir que compressores atuais como WinZip e WinRar utilizam esses algoritmos nos interessamos a conhecer melhor seu funcionamento, e através de técnicas de análise de desempenho, como a taxa de compactação, velocidade de compactação e uso da memória, descobrir o melhor algoritmo para cada tipo de arquivo. Utilizaremos uma tabela de arquivos padrão para análise de desempenho de algoritmos conhecida como CanterBury Corpus, e faremos a comparação dos seus

resultados gerais a fim de determinar o melhor entre o algoritmo de Huffman e o algoritmo de LZW. Para isso tivemos a idéia de desenvolver um protótipo ferramenta utilizando uma tecnologia atual o JAVA, onde programaremos os dois algoritmos e programaremos também técnicas de análise de desempenho, como a taxa de compactação, velocidade de compactação e uso da memória

Com o desenvolvimento desta ferramenta visamos obter os algoritmos mais precisos para os tipos de arquivo propostos e com isso obter uma melhora no desempenho, na transferência de dados desse tipo de informação, sem que haja perigo de extravio ou perda de informação. A partir dos resultados visamos buscar entre os dois algoritmos o melhor algoritmo para cada tipo de entrada.

1.4 ESTRUTURA DO TRABALHO

O presente trabalho é constituído de cinco capítulos: Introdução; Compactadores; algoritmo de Huffman; algoritmo de LEMPEL -ZIV & WELCH LZW, ferramenta proposta, análise de desempenho, trabalhos correlatos e conclusão.

A Introdução é constituída de objetivo geral, objetivos específicos, justificativa e a estrutura do trabalho.

O capitulo 2 apresenta informações sobre compactadores, funcionamento e definições de grande importância.

O capitulo 3 apresenta informações sobre o algoritmo de Huffman, funcionamento e codificação.

O capitulo 4 apresenta informações sobre o funcionamento sobre o algoritmo de LEMPEL-ZIV & WELCH LZW.

O capítulo 5 apresenta a ferramenta proposta e definições de funcionamento.

O capitulo 6 apresenta informações sobre análise de desempenho, e os resultados obtidos pela ferramenta em relação aos algoritmos.

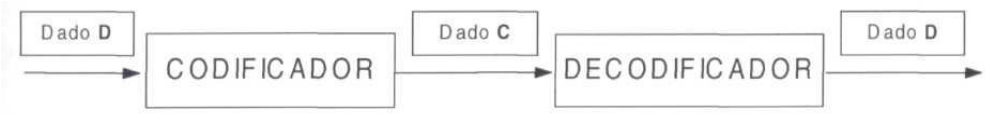
O capitulo 7 apresenta os trabalhos correlatos.

O capitulo 8 apresenta as conclusões obtidas através deste trabalho de pesquisa.

2 COMPACTAÇÃO DE DADOS

De acordo com FRANÇA NETO (1998) a compactação de dados tem como intuito tentar reduzir ao máximo o espaço ocupado em um dispositivo de armazenamento de dados, por exemplo, discos rígidos ou pendrives. Esse tipo de operação é feita através de algoritmos de compactação de dados no qual o objetivo é reduzir a quantidade de bytes utilizados para representar determinados tipos de arquivos. Essa compactação de dados tem como objetivo reduzir a redundância de informação utilizada para representar um arquivo. Na figura 1 temos um esquema que FRANÇA NETO (1998) utilizou para representar a compactação e descompactação de determinado arquivo “D” onde este arquivo é comprimido se tornando o arquivo “C” e depois descomprimido voltando a ser o arquivo “D”.

Figura 1- Esquema de compactação.



Fonte: Compactação de dados: Técnicas e aplicações, considerações de Hardware e Software. HELD (1992) 19p

Ainda de acordo com o mesmo autor essa redução é feita através de algoritmos que eliminando os bits redundantes na informação diminuem o tamanho dos arquivos. Um exemplo comum dessa compactação é dado através da seqüência “XXXXXXX” que ocupa sete bytes e que poderia ser representada através do código “7X” onde foi aplicada a técnica de Run-length encoding (RLE) que faz a supressão de caracteres repetidos do código segundo estudos de HELD (1992). Esse código em si ocupa bem menos bytes, neste caso dois, assim economizando cerca de 68% de espaço de armazenamento.

As compressões de dados além de eliminar redundâncias em códigos tem como objetivo principal economizar mais espaço de armazenamento em unidades de disco, desta forma essas unidades ganham desempenho tanto no armazenamento quanto na transmissão de dados.

A compressão e a compactação são dois processos diferentes, a compressão visa reduzir a quantidade de bits utilizados para representar uma

informação, a compactação serve para agrupar os dados que não estejam agrupados.

A partir das definições HELD (1992) podemos dizer que atualmente a compactação de dados está presente nas aplicações de comunicação, como redes de computadores, onde ela é utilizada em protocolos de comunicação para reduzir as informações na hora de enviá-las. Vários padrões de compactação estão presentes no nosso dia-dia, tais como o ZIP, RAR entre outros. A compactação de dados possui inúmeras aplicações podendo ser utilizada tanto no nível de hardware quanto no nível de software.

Dentre as técnicas de compactação de dados podemos classificá-las em compactação com perda e compactação sem perda.

HELD (1992) afirma que algoritmos de compactação com perda são conhecidos como quantizadores, neste tipo de compactação o dado original é processado através de um método de quantização, permitindo altas taxas de redução do tamanho original. Porém alta taxa de redução de tamanho tem um custo, que seria a perda de informação em relação ao dado original, o arquivo perde a fidelidade em relação ao arquivo original.

Seguindo as afirmações de HELD (1992) podemos dizer que algoritmos de compactação sem perda são conhecidos como codificadores universais, esse tipo de codificador mantém a fidelidade do arquivo em relação ao arquivo de origem. Porém essa fidelidade tem um custo, em que as taxas de redução são diminuídas, ou seja, quando utilizamos compressores que utilizam métodos sem perda obtemos taxas de compactação menores, os arquivos compactados possuem tamanho modestamente menor. Isso se dá devido ao limite entrópico, o limite entrópico é a quantidade de caracteres diferentes presentes dentro da cadeia de símbolos, a entropia pode ser caracterizada como a quantidade real de dados de uma informação existente.

HELD (1992) exemplifica o funcionamento do método de compactação sem perda através de um arquivo de texto plano, como esse arquivo de texto possui uma entropia bem menor que o espaço que ocupa armazenado em bytes, textos comuns possuem grande redundância de informações, que está associada há má organização da informação.

Seguindo as afirmações de HELD (1992) podemos dizer que a compactação de dados é o processo de codificar um grupo de informações dentro de uma estrutura menor em relação à original, sendo possível uma reconstrução para o tamanho original depois. Dentro disso a compactação de dados é basicamente receber uma seqüência de símbolos e transformá-la em uma seqüência de códigos. Dependendo da eficiência da compactação o código resultante será menor que a seqüência original. Na figura 2 temos o esquema de compactação de dados descrito na obra de HELD.

Figura 2: Diagrama de blocos de compactação de dados. Um fluxo de dados originais operando de acordo comum ou mais algoritmos de compactação, resulta na geração de um fluxo de dados compactados (Held 1992)

Fonte: Compactação de dados: Técnicas e aplicações, considerações de Hardware e Software. HELD (1992) 25p

Segundo HELD (1992) a compactação de dados é muito útil em vários casos, podemos citar dentre esses casos a manutenção de dados on-line, a redução de tempo necessário para a transferência de dados, o retardo na aquisição de mais discos de dados que também são conhecidos como HDs, redução do número de fitas back-up, pois grandes servidores de marcas com DELL e HP ainda possuem back-up em fitas.

Seguindo as afirmações de HELD (1992) existem algumas técnicas que são comumente utilizadas dentro da compactação de dados, dentre elas podemos citar a Supressão de Repetições (Run Lenght) também conhecida como codificação de comprimento de fileira.

HELD (1992, 398p) apud Rubin (1976), Ruth e Kreutzer (1972) esclarece que:

> “A codificação de comprimento de fileira é um método de compactação de dados que reduzirá, fisicamente qualquer tipo de seqüência de caracteres repelidos, desde que a seqüência de caracteres alcance um nível predefinido de ocorrência. Para a situação especial onde o caractere nulo é o caractere repetido a compactação de comprimento de fileira pode ser vista como um processo avançado de supressão de nulos.”

Com base nos estudos dos autores FRANÇA NETO (1998) e HELD (1992), neste trabalho demos foco a supressão de repetições que é a técnica a ser empregada pelos algoritmos escolhidos para serem utilizados no nosso projeto.

Na supressão de repetições os dados a serem compactados podem ser códigos executáveis ou textos de dados, nestes casos podem ter várias seqüências de repetição de caracteres, podemos constatar que a ocorrência de X repetições do caractere Y podem ser substituídos por “n” bytes.

Cameron Buschardt (1997) em seu artigo “File Compression” descreve de forma resumida o Run Lenght como um algoritmo baseado na teoria de converter grandes cadeias de caracteres dispostos de forma repetitiva como no exemplo ‘aaaaaaaaabbbbbbbcccc’ em seqüências comprimidas que não utilizam caracteres especiais. Essa seqüência seria codificada da seguinte forma 9a7b4c, onde a letra 'a' ocorreu 9 vezes, o 'b' 7 vezes e o ' c ' 4 em relação à seqüência original.

SZWARCFITER & MARKENZON (1994) afirmam que se o texto a ser comprimido tiver caracteres numéricos, um código ambíguo seria gerado se for codificado somente desta forma, pois os números do texto seriam confundidos com os números que indicam freqüência de repetições. Para resolver este problema, geralmente colocam-se símbolos que não tenham na fonte de entrada, por exemplo, @, para anunciar que o número imediatamente após representa um caractere numérico e não um símbolo de freqüência. Assim a cadeia anterior seria codificada da seguinte maneira @9a@7b@4c.

De acordo com SZWARCFITER & MARKENZON (1994) podemos afirmar que todas as técnicas de compactação têm o objetivo de diminuir o comprimento de um código buscando um código reduzido ótimo. Para analisarmos a eficiência destas diferentes técnicas usamos uma métrica chamada de “taxa de compactação” que visa descobrir o percentual de redução entre o arquivo compactado e o arquivo original, conforme a figura 3.

Figura 3. Fórmula para cálculo da taxa de compactação/compactação.

$$Taxa\ de\ compressão = \left(1 - \frac{Tamanho\ Comprimido}{Tamanho\ Original}\right) \times 100$$

Fonte: Scribd cursos de tecnologia na informação. Artigo sobre compactação publicado em www.scribd.com.br

Essa medida está relacionada entre o tamanho da entrada menos o tamanho da saída dividido pelo tamanho da entrada, essa métrica é demonstrada através de percentual que indica o quanto o arquivo foi reduzido em relação ao arquivo de origem, também chamado nesse caso de arquivo de entrada, a taxa de compactação será um dos aspectos abordados nesse trabalho.

2.1 COMPACTAÇÃO SEM PERDA

BUSCHART (1997) afirma em seu estudo “File Compression” que a compactação de dados sem perda é o método aplicado por determinados algoritmos, em que a informação após ser compactada e descompactada permanece idêntida a informação original de entrada. Esse método é constantemente usado na compactação de imagens, onde após a compactação a imagen continua com o mesmo número de pixels mas reduz o espaço de armazenamento em bits.

Existem diferentes tipos de algoritmos de compactação sem perda, sendo os mais conhecidos e importantes o algoritimo de Huffman e o algoritmo LZW.

Figura 4 - Esquema do algoritmo de Huffman

$n \leftarrow |C|;$
$Q \leftarrow C;$
para $i \leftarrow 1$ até $n-1$ faça
    $CriaNo(z);$
    $x \leftarrow z.esq \leftarrow ExtraiMinimo(Q);$
    $y \leftarrow z.dir \leftarrow ExtraiMinimo(Q);$
    $f[z] \leftarrow f[x] + f[y];$
    $Insere(Q, z);$
retorne $ExtraiMinimo(Q);$

Fonte: Compactação de Arquivos e Algoritmo de Huffman, UNICAMP, Artigo Publicado em www.unicamp.com.br

Ainda seguindo as definições de BUSCHART (1997) podemos dizer que a compactação sem perda é indispensável em várias aplicações onde o grau de segurança deve ser elevado e também nas aplicaçoes onde a fidelidade das informações é estritamente necessária. Imagens médicas são um bom exemplo desta aplicação, essas imagens não podem ter nenhum tipo de informação perdida durante a compactação e descompactação pois essa perda pode provocar o

diagnóstico equivocado de um determinado paciente, resultando em um erro grave de avaliaçao médica, nestes casos não poderiam ser utilizadas técnicas de comprensão com perda.

Esse método de compactação é bastante utilizado em aplicações, um exemplo popular seriam os arquivos ZIP para o windows e os arquivos gZIP utilizados no unix. Esse método também é utilizado em arquivos de imagem, um exemplo comum são os arquivos PNG onde apenas o método de compactação de dados sem perda é utilizado enquanto arquivos de outras extensões com TIFF e MNG podem utilizar tanto compactação sem perda como compactação com perda, dependendo da maneira com for implementado.

Com base nos estudos de FRANÇA NETO (1998) podemos dizer que as imagens de extensão GIF também utilizam os métodos de compactação sem perda, mais neste caso possuem uma particularidade pois nenhuma das implementações GIF são capazes de representar as cores, para que eles possam ser reduzidos de forma a não perder informação. Para isso são utilizados métodos de quantização de cores, este método gera perdas, mais ao reconstruir a imagem são utilizados processos de re-quantização de cores e desta forma não há perda adicional. Nos estudos de FRANÇA NETO (1998, 136p) ele afirma:

> "Uma das aplicações, é a variação implementada no formato GIF, utilizando no armazenamento de imagens. Atualmente, este é um dos formatos de armazenamento de imagens sem perdas que oferece as melhores taxas de compactação. Também encontrado em outros formatos gráficos"

2.1.1 Técnicas de compactação sem perdas

BUSCHART (1997) afirma que as técnicas utilizadas em algoritmos de compactação sem perda podem ser classificadas de acordo com o tipo de arquivo que se deseja comprimir. Os principais tipos de arquivo a serem compactados pelos algoritmos de compactação são textos, executáveis e imagens. Independente disso podemos dizer que algoritmos de compactação de dados sem perda podem ser utilizados em qualquer tipo de arquivo, embora dependendo da forma como foram implementadas podem ser inúteis para determinados tipos de dados por não atingirem uma taxa significativa de compactação. HELD (1992) ainda afirma que as aplicações que utilizam compactação sem perda podem usar dois tipos de técnicas

dentro da sua estrutura, a primeira técnica gera um modelo estatístico para uma determinada entrada de dados e a outra técnica mapeia a entrada dos dados para sequências de bits.

HELD (1992, 300p), em sua obra define a codificação estatística através da seguinte exemplificação:

> "O processo de codificação estatística pode ser usado para obter uma minimização do comprimento médio dos dados codificados de uma forma semelhante àquela no qual o Morse selecionava as representações por ponto e traço para os caracteres, de maneira que um único ponto era usado para representar a letra E, que é o caractere mais freqüentemente encontrado na língua inglesa, enquanto grandes cadeias de pontos e traços foram usados para representar caracteres que aparecem com menos freqüência,"

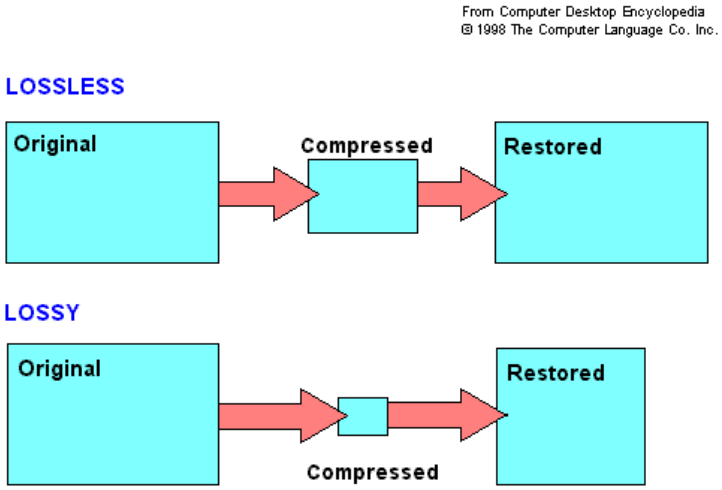
Geralmente a técnica estatística é mais utilizado durante a execução da aplicação, enquanto a segunda técnica que mapeia a entrada dos dados para sequências de bits está dentro do código, mais geralmente não é utilizado pois o modelo estatístico segundo HELD (1992) é mais eficaz. Um exemplo de algoritimo que utiliza a técnica estatística é o algoritimo de LEMPEL-ZIV & WELCH LZW que trataremos mais a frente no capítulo 4, um exemplo de algoritmo que utiliza a técnica de sequência de bits é o algoritmo de Huffman que trataremos mais a fundo no capítulo 3.

2.2 COMPACTAÇÃO COM PERDA

BUSCHART (1997) afirma que o método de compactação com perda possui uma vantagem sobre o método sem perdas, que seria uma maior taxa de compactação à medida que perde dados durante o processo, podendo gerar arquivos muito menores em relação ao outro método. Na maioria dos casos geram arquivos tão pequenos que nenhuma técnica de compactação sem perda consegue o mesmo resultado.

Na figura 5 temos uma representação da compactação sem perda e outra da compactação com perda.

Figura 5: A figura representa a diferença entre os arquivos compactados utilizando as técnicas de compactação com perda e sem perda



Fonte: Computer Desktop Encyclopedia, 1994. Publicado em www.CDE.com.br

Segundo FRANÇA NETO (1998) os métodos de compactação com perda estão presentes em diversas aplicações e são utilizados quando a fidelidade do arquivo de saída não precisa ser mantida em relação ao arquivo de entrada, exemplo disso são arquivos de som, onde os algoritmos de compactação com perda retiram do som apenas faixas inaldiveis aos seres humanos, não causando uma perda significativa e diminuindo o tamanho do arquivo. Podemos analisar as taxas de compactação desses tipos de arquivos utilizando a fórmula expressa na figura 3.

Em seu artigo “Compactação de Dados” MULLER (2010) afirma após a realizações de testes que um arquivo de som pode ser comprimido a uma taxa de compactação 10:1, essa taxa de compactação é obtida em relacionando o tamanho oiginal do arquivo com o tamanho do arquivo compactado, sem uma perda significativa de qualidade com um algoritmo de compactação com perda. Um arquivo de vídeo consegue uma taxa de compactação de até 300:1 sem que haja perda de qualidade perceptível em relação ao arquivo original qualidade com um algoritmo de compactação com perda. Em arquivos de imagem a compactação chega a 1/1000 em relação ao tamanho do arquivo original de entrada, mais nesse caso a perda de dados é perceptivel fazendo uma análise mais detalhada do arquivo, sendo que em alguns casos essa perda pode ser vista também em arquivos de audio.

De acordo com as afirmações de HELD (1992) é possível dizer que um arquivo qualquer comprimido, utilizando a compactação com perda para um determinado fim, exemplo armazenamento ou até mesmo transmissão em uma rede, ao ser descomprimido pode conter aspectos bem diferentes do arquivo original, perdendo assim a fidelidade das informações contidas. Geralmente essas diferenças nos aspectos dos arquivos que utilizam essa técnica não são significativas sendo impossíveis de serem detectadas sem uma análise minusiosa.

Muitos desses métodos de compactação sem perda utilizam-se da falta de capacidade do corpo humano de reconhecer certas partes das informações sejam elas visuais, exemplo imagens e vídeos, ou sonoras como sons e arquivos de audio. Desta forma esses métodos ou técnicas retiram dos arquivos aquilo que não podemos captar, sem o uso de equipamentos específicos de análise. Um exemplo desta remoção são as frequências sonoras que ocorrem em arquivos MP3, em que são retiradas frequências inaudíveis aos seres humanos.

Podemos então dizer que quando ocorrem perdas significativas de informação durante a compactação com perda e essas se tornam visíveis aos seres humanos, sem o uso de nenhum equipamento especial, chamamos de “Artefatos de Compactação”. Artefatos de compactação ocorrem quando são alcançadas altas taxas de compactação, quanto mais se comprime um arquivo maior a chance de ocorrerem defeitos no arquivo descomprimido. Esses defeitos ocorrem, pois os algoritmos de compactação, geralmente utilizados para comprimir os arquivos de vídeo, acarretam perda de informações durante a compactação e não temos como descomprimi-los no processo. Esses defeitos nos dados são apresentados em erros nas corres ou na estrutura da resolução em partes da imagem.

HELD (1992) afirma que os algoritmos de compactação sem perda de dados geralmente exploraram a redundância estatística dentro dos arquivos a serem compactados, de forma a representar de forma mais correta e fiel a informação mesmo após a descompactação. Desta forma a compactação sem perdas pode ser utilizada por que na maior parte da informação transitada e utilizada nos dias atuais existem redundâncias estatísticas utilizando cálculos de probabilidade em sua definição.

2.3 CLASSIFICAÇÕES DOS MÉTODOS DE COMPRESSÕES DE DADOS

Segundo HELD (1992) ainda dentro da compactação de dados tanto com perda quanto na compactação sem perda, temos algumas classificações a serem definidas, dentro dessas classificações temos a simetria, a adaptabilidade de fluxo, bloco e operações. Essas classificações definem de forma especifica o comportamento da compactação de dados.

2.3.1 Classificação por Simetria

HELD (1992) afirma que dentro da compactação de dados temos os métodos simétricos e métodos assimétricos, esses métodos na verdade são as diferenças entre a compactação e descompactação. Dizemos que um método de compactação é simétrico quando o algoritmo de compactação e descompactação são iguais ou bem parecidos. Exemplos bem conhecidos de compactação simétrica são os algoritmos LZW e os de codificação aritmética. Nesses dois métodos os algoritmos são praticamente iguais e tem a mesma complexidade de desenvolvimento.

Ainda segundo HELD (1992) podemos dizer que um método de compactação é chamado de assimétrico quando o algoritmo de compactação é mais complexo que o de descompactação. Essa complexidade implica tanto no tempo de desenvolvimento quanto no tempo de execução. Esse tipo compactação é muito utilizado quando comprimimos um determinado arquivo uma vez e o descomprimimos muitas vezes, por exemplo, arquivos de música ou sons em geral. Nesses casos a compactação pode levar mais tempo, mais sua descompactação é obtida de forma mais rápida. Exemplos de algoritmos assimétricos é o DEFLATE e o LZ77 que é uma derivação do LZW que estudaremos mais a frente.

2.3.2 Classificação por Adaptabilidade

De acordo HELD (1992) a adaptabilidade dos algoritmos de compactação em geral podem ser classificados como adaptáveis e não-adaptáveis. Os algoritmos não-adaptáveis seguem regras rígidas de tratamento de dados que não dependem nem do tipo de informação a ser comprimido e nem mesmo das alterações enquanto os dados são lidos. Em contrapartida existem os métodos

adaptáveis aonde o algoritmo de compactação vai se adaptando de acordo com que os dados vão sendo processados, alguns exemplos de métodos adaptativos são os algoritmos LZ77 e LZ78.

Os métodos de codificação de Huffman ou codificação aritmética são exemplos de algoritmos que podem ser classificados das duas formas, tanto adaptativo quando não-adaptativo, um exemplo de código não adaptativo é o código de Golomb.

2.3.3 Classificação por Fluxo e Bloco

Segundo afirmações de HELD (1992), normalmente métodos de compactação de dados utilizam fluxo contínuo de informação, buscando de certa forma a transmissão e o armazenamento seqüencial dessa informação. Métodos mais novos de compactação aproveitam a proximidade dos dados uns dos outros para poder agrupar e processar em forma de conjuntos, desta maneira visando aumentar as taxas de compactação do arquivo. Um exemplo desta classificação é o método de compactação Burrows-Wheeler.

2.3.4 Classificação por Operação

HELD (1992) afirma que na classificação por tipo de operação os algoritmos são classificados de acordo com o objetivo que buscam atingir, sua finalidade ou tipo de resultado de compactação que se deseja. Dentro dessa classificação podemos citar os métodos, “Estatístico”, “Redução de Redundância” e o método de “Transformações”.

Ainda segundo HELD (1992) os métodos estatísticos utilizam a probabilidade como forma de prever a ocorrência dos mesmos símbolos dentro de uma determinada seqüência de informações. Sabendo o número de ocorrências de determinados símbolos ele busca reduzir o número de bits utilizados para representar cada um deles. Esses métodos estão diretamente ligados à teoria da informação e como exemplo podemos citar o código de Huffman. Métodos de redução de redundância utilizam dicionários de dados ou estruturas parecidas a fim de eliminar a ocorrência de símbolos iguais dentro da mesma estrutura, exemplos de métodos que utilizam a redução de redundância são o LZ77 e o LZ78, esses

métodos aliam as técnicas de estatística e probabilidade a técnica de redução de redundância.

Métodos de transformações não são métodos de compactação, na realidade são métodos que transformam informações que não poderiam ser comprimidas ou sua compactação não teria como decorrência um resultado eficiente. Após a aplicação do método de transformação esses dados poderão ser mais facilmente compactados. Esse tipo de técnica é utilizada pela compactação de dados com perda, geralmente utilizado para eliminar a relação entre dados adjacentes, desta forma eliminando informações sem prejuízo no resultado final.

3 ALGORITMO DE HUFFMAN

O algoritmo de Huffman é segundo FRANÇA NETO (1998) um dos algoritmos de compactação sem perda mais conhecido, inicialmente este método foi aplicado à compactação de arquivos de texto onde buscava obter uma taxa de compactação ótima. David Huffman desenvolveu esta técnica que visava utilizar árvores binárias a fim de gerarem códigos binários. Dentro desta técnica se suponha que o arquivo de texto a ser compactado fosse composto por uma seqüência de caracteres ou símbolos, conhecendo a freqüência com que cada símbolo aparece seria possível atribuir um valor para cada símbolo através do número de ocorrências, desta forma compactando o arquivo inteiro. FRANÇA NETO (1998, 67p) afirma em sua obra que, "O algoritmo de Huffman é um aperfeiçoamento do algoritmo de Shannon-Fano, desenvolvido em 1952 por David Huffman. Desde então, surgiram diversas variações do algoritmo", apenas o algoritmo de Huffman foi desenvolvido por David Huffman, o Shannon-Fano foi desenvolvido por por Claude Shannon e Robert Fano.

Seguindo as afirmações do autor o algoritmo de Huffman revolucionou a compactação quando foi criado, se tornou um dos algoritmos mais conhecidos sendo implementado por universidades pelo mundo todo. Seguindo a mesma lógica do algoritmo de compactação de Shannon-Fano o algoritmo de Huffman tem como objetivo principal a codificação dos símbolos que aparecem com uma freqüência maior com um número menor de bits e aqueles que aparecem com menos freqüência com um número maior de bits. Com isso podemos afirmar que as taxas de compactação através desse algoritmo podem ser maiores ou menores dependendo da probabilidade da distribuição dos símbolos dentro de um arquivo. A compactação através deste algoritmo é dada através de dois passos, no primeiro determinamos a forma que a probabilidade é distribuída entre os símbolos da fonte, utilizamos essa distribuição para a geração da tabela de códigos. No segundo passo é feita uma varredura na tabela gerada anteriormente, desta forma determinamos a codificação e as fontes reduzidas, apões determinarmos as codificações a serem utilizadas o arquivo por fim é compactado.

Segundo FRANÇA NETO (1998) a diferença entre o algoritmo de Shannon-Fano e o de Huffman está basicamente na maneira em que a árvore binária é construída, a árvore de decisão de Huffman não é gerada baseando-se em divisões de dois grupos de símbolos que é utilizada pelo algoritmo de Shannon-Fano, no qual a soma das probabilidades tem que ser igual ou semelhante, mais sim na soma das menores freqüências de símbolos encontradas nos arquivos de entrada.

Figura 6: Algoritmo de Huffman

$n \leftarrow |C|;$
$Q \leftarrow C;$
para $i \leftarrow 1$ até $n-1$ faça
    $CriaNo(z);$
    $x \leftarrow z.esq \leftarrow ExtraiMinimo(Q);$
    $y \leftarrow z.dir \leftarrow ExtraiMinimo(Q);$
    $f[z] \leftarrow f[x] + f[y];$
    $Insere(Q, z);$
retorne $ExtraiMinimo(Q);$

FONTE: Artigo de compactação de arquivos publicado por UNICAMP.

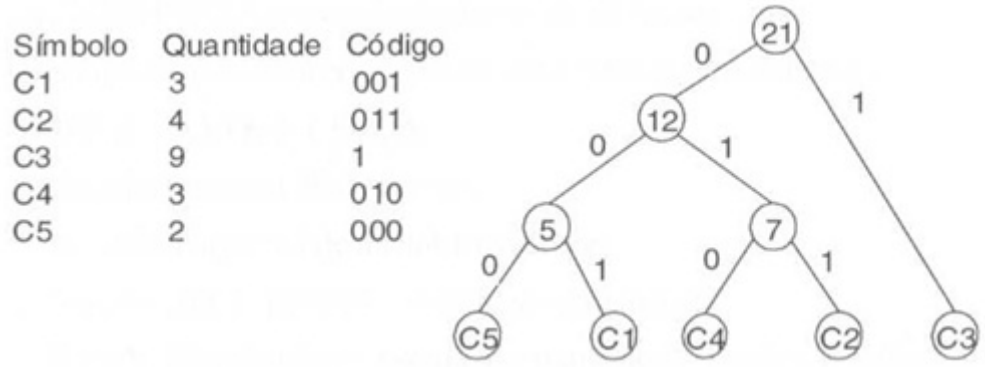
3.1 SÍNTESE DO ALGORITMO

FRANÇA NETO (1998) afirma que a técnica de compactação utilizada pelo algoritmo de Huffman tem como objetivo executar a representação em binário a partir da probabilidade de ocorrência de caracteres, a representação é gerada por algoritmos de decodificação através de árvores binária que impossibilita haver ambigüidade na análise do código.

Ainda seguindo o autor pode-se afirmar que neste caso a ambigüidade citada é quando existe confusão com outros caracteres durante a decodificação, um exemplo disso pode ser dado apresentando um determinado caracter Cl que possui código binário 01 e um caracter C2 que possui código 0100, isto define que, ao analisarmos o código binário para C2 poderemos estar interpretando como C1, ao serem lidos apenas os dois primeiros bits, 01. Para evitar isso à codificação de Huffman faz uso de árvores binárias para a estrutura dos bits utilizados para a representação de caracteres, desta forma permitindo uma decodificação individual

para cada caracter este processo é conhecido como árvore de Huffman, conforme ilustra a figura 7.

Figura 7: Esquema da arvore de Huffman.



Fonte: Adaptado de FRANÇA NETO (1998).

A técnica de compactação criada por Huffman é explanada por vários autores, desta forma demonstraremos a utilização desse método segundo a abordagem sugerida por FRANÇA NETO (1998). Esquema sugerido por FRANÇA NETO (1998) é apresentado na figura 8.

Figura 8: Esquema da árvore de Huffman

1. Os símbolos devem ser colocados em seqüência de probabilidade decrescente.

2. Juntam-se os dois símbolos de probabilidade menor com a junção desses símbolos geramos um novo símbolo no qual a probabilidade será a soma das probabilidades dois símbolos utilizados na união. A partir desses símbolos criamos uma nova fonte e essa fonte chamamos de fonte reduzida, essa fonte terá um numero menor de elementos em relação a fonte original, mantemos a seqüência sempre juntando dois símbolos até o fim do arquivo e gerarmos ele na sua forma reduzida.

3. Neste passo se mantém o funcionamento do passo anterior, gerando novas fontes reduzidas, com objetivo de chegar a uma seqüência onde restem apenas dois símbolos.

4. No passo quatro utilizamos os dois símbolos que sobraram, utilizando-os para fazer uma atribuição aleatória de 1 para um dos símbolos e 0 para o outro símbolo, essa atribuição não é uma regra, a implementação disso fica a critério do desenvolvedor.

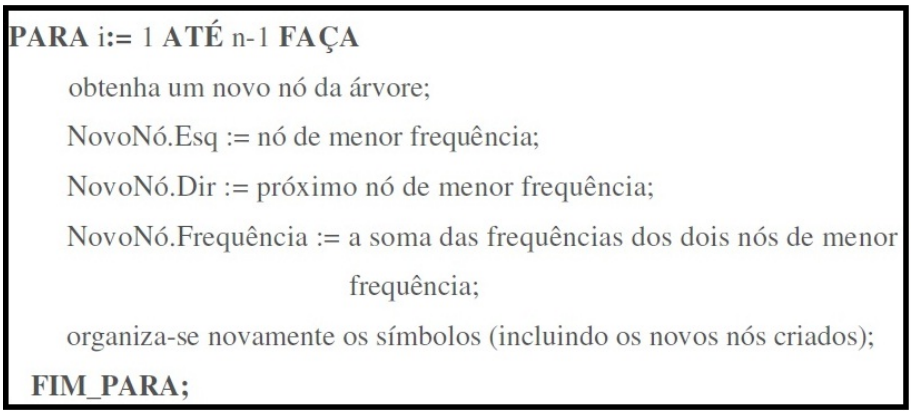
5. Continuado com a execução do algoritmo, voltasse para o inicio da fonte atribuindo zeros e uns para as palavras do arquivo.

6. Ao chegarmos nos símbolos iniciais da fonte a codificação é finalizada e o arquivo esta pronto para ser utilizado.

Fonte: Adaptado de FRANÇA NETO (1998).

FRANÇA NETO (1998) afirma ainda em seus estudos que no decorrer da codificação de Huffman a distribuição de zeros e uns é feita aleatoriamente, como mostrado no passo quatro deste algoritmo, desta forma tornando viável a construção de uma variedade de diferentes códigos, esses códigos possuíram o mesmo comprimento. Este algoritmo também pode ser representado em pseudocódigo como na figura 9.

Figura 9: Algoritmo para determinação de decodificação de Huffman.



Fonte: Adaptado de FRANÇA NETO (1998).

Seguindo a lógica sugerida por TERADA (1991) e SZWARCF1TER & MARKENZON (1994) o algoritmo pode ser representado pela complexidade O (n log“n”). Nesta linha podemos dizer que criando os nós da árvore binária de forma que os símbolos sejam codificados utilizando a menor seqüência de zeros e uns possíveis, chegamos a árvore binária de Huffman ou árvore de Huffman.

Para exemplificar o algoritmo Huffman faremos a codificação dele utilizando um alfabeto bastante conhecido sugerido por SZWZRCFITER (1994). Para a codificação de Huffman necessitamos que cada símbolo possua um valor para probabilidade de ocorrência, ou seja, o percentual relacionado ao número de vezes que esse símbolo aparece na entrada. Iniciam-se a construção da árvore binária de Huffman a partir dos caracteres que possuem valor menor dentro do arquivo, ou alfabeto neste caso. Como exemplo, podemos demonstrar na figura 10 a distribuição do percentual aos elementos já ordenados.

Figura 10: Caracteres ordenados pela sua probabilidade.

Caracter	Probabilidade	Quantidade
C3	0,44	9
C2	0,19	4
C4	0,14	3
C1	0,14	3
C5	0,09	2

Fonte: Adaptado de SZWZRCFITER (1994).

Primeiro passo:

Inicialmente juntamos os dois caracteres de probabilidade menor de ocorrência em um nó que possuirá a uma freqüência, essa freqüência será a junção das probabilidades dois caracteres que foram unidos, neste caso seria a soma das probabilidades 0,09 e 0,14 e essa soma resultara na freqüência de 0,23. Para uma melhor visualização gráfica e entendimento representaremos a freqüência utilizando a soma das quantidades de ocorrências de determinado símbolo, somamos neste caso as quantidades 2 e 3 obtendo o resultado de 5, em primeiro momento.

Segundo passo:

A seguir continuamos com a execução, buscamos o par de caracteres com freqüência menor, neste caso temos o C4 e o C2, juntaremos esses dois símbolos em um novo nó que a freqüência será a soma das freqüências dos dois símbolos, neste caso a soma entre 3 e 4 resultando em um nó de freqüência 7.

Terceiro passo:

Após o primeiro e o segundo passo já formamos os dois ramos com as menores freqüências encontradas que são o 5 e o 7, juntamos esses dois nós em um ramo de freqüência 12, que é a soma das freqüências dos nós que agrupamos.

Quarto passo:

No quarto e ultimo passo utilizamos o caractere C3, o adicionamos a arvore gerando um nó pai cuja freqüência será de 21 que é o total de caracteres da fonte, este será a raiz da arvore binária, em seguida atribuímos zero ao nó filho da raiz à esquerda e um ao nó filho da direita, neste caso a atribuição pode ser inversa, a regra não exige que seja zero a direita e um a esquerda, tomamos isso só por definição.

Para exemplificar o funcionamento do algoritmo de Huffman utilizaremos a string “bom esse bombom”, onde os caracteres ‘b’, ‘o’ e ‘m’ têm peso 3, pois aparecem 3 vezes na string, os caracteres ‘e’, ‘s’ e espaço têm peso 2. A árvore criada pelo algoritmo de Huffman será construída a partir de um grupo de árvores. Inicialmente essas árvores têm como um único nó com um caractere e o peso deste caractere. A cada iteração do algoritmo, duas arvores são juntadas criando uma nova árvore. Isso faz com que o numero de árvores diminua a cada passo, o algoritmo pode ser observado na figura 11.

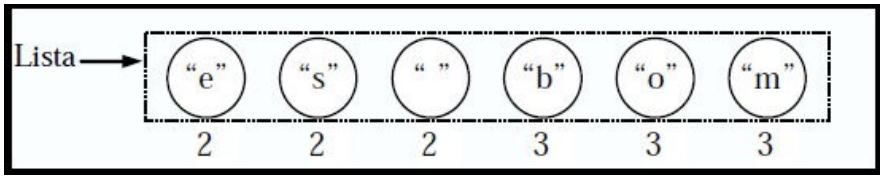
Figura 11: Esquema do algoritmo de Huffman.

1. Comece com uma lista de árvores. Inicialmente, todas as árvores são compostas de um nó apenas, com o peso da árvore igual ao peso do caracter do nó. Caracteres que ocorrerem mais frequentemente têm o maior peso. Caracteres que ocorrerem menos frequentemente têm o menor peso. Ordene a lista de árvores de forma crescente, fazendo com que o nó raiz da primeira árvore seja o caracter de menor peso e o nó raiz da última árvore seja o caracter de maior peso.
2. Repita os passos a seguir até que sobre apenas uma única árvore:
    - Pegue as duas primeiras árvores da lista e as chame de T1 e T2. Crie uma nova árvore Tr cuja raiz tenha o peso igual à soma dos pesos de T1 e T2 e cuja subárvore esquerda seja T1 e subárvore direita seja T2.
    - Exclua T1 e T2 da lista (mantendo T1 e T2 na memória) e inclua Tr na lista,de maneira que a lista seja mantida ordenada.
3. A árvore final será a árvore ótima de codificação.

Fonte: Material didático utilizado pela professora Christine Vieira Scarpato.

Na figura 12 mostraremos a fase inicial do algoritmo para string “bom esse bombom”, mostraremos os nós com os respectivos pesos que demonstram o numero de vezes que o respectivo caractere aparece na string de entrada, a lista é ordenada conforme o peso de cada caractere presente nela.

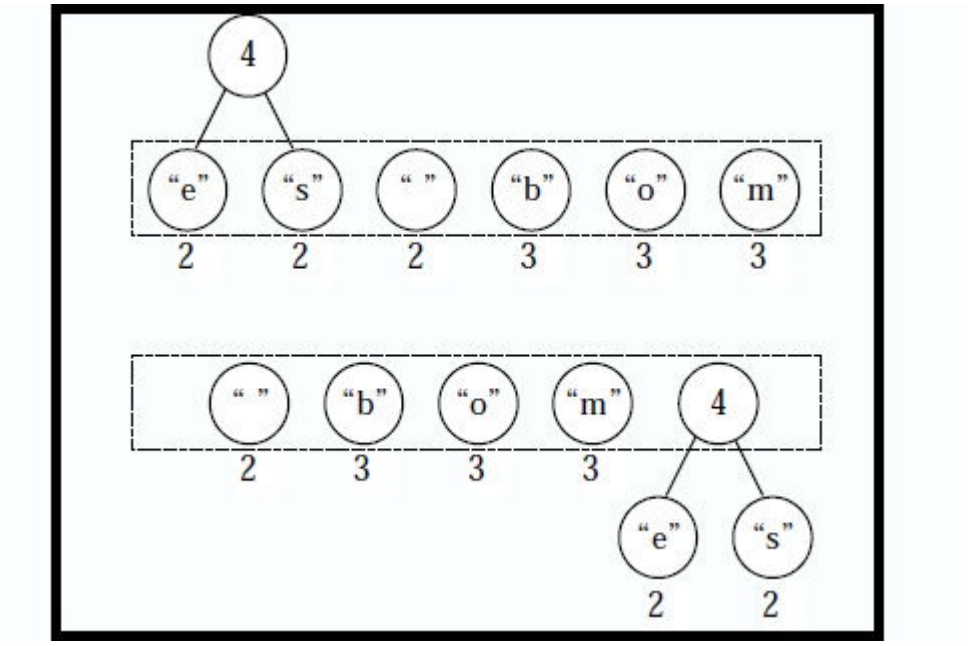
Figura 12: Etapa inicial do algoritmo de Huffman, após passo 1



Fonte: Material didático utilizado pela professora Christine Vieira Scarpato.

Os dois primeiros nós são juntados para formar uma nova árvore, cujo peso do nó raiz é a soma dos nós dos pesos dos nós filhos que podemos observar na figura 13, continuando na execução do algoritmo, os dois primeiros nós são retirados da lista e o nó raiz da arvore criada é inserido ordenadamente na lista de arvores. A figura 13 mostra à nova arvore sendo inserida no final da lista.

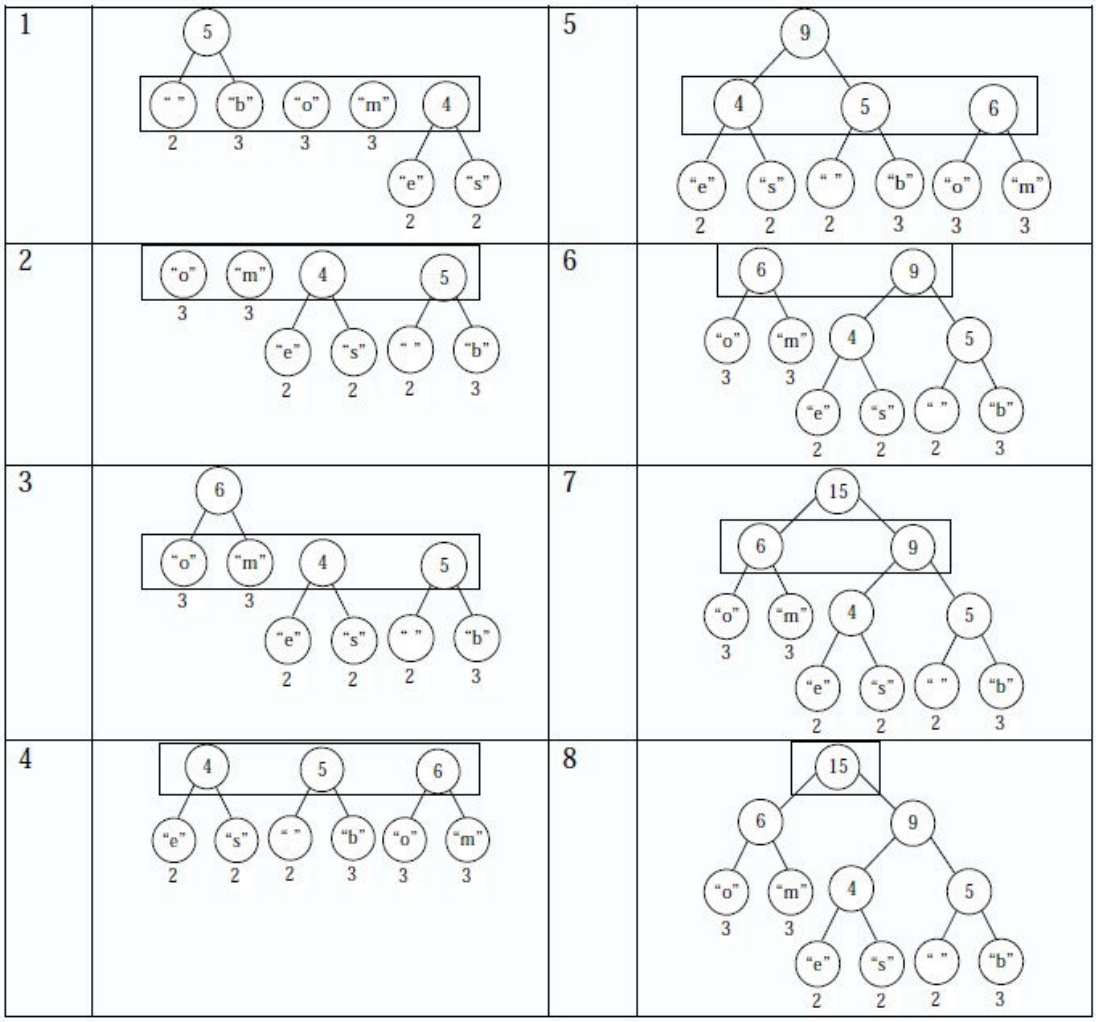
Figura 13: Após passo 2 do algoritmo executado pela primeira vez



Fonte: Material didático utilizado pela professora Christine Vieira Scarpato.

Repetimos os passos acima até que sobre apenas uma árvore, figura 14 a seguir demonstra o resto da execução do algoritmo passo-a-passo.

Figura 14: Esquema passo-a-passo da execução do algoritmo.



Fonte: Material didático utilizado pela professora Christine Vieira Scarpato.

A codificação de caracteres induzidas pela arvore 8 é a presente na tabela 1 levando em consideração 0 para arcos à esquerda e 1 para arcos à direita.

Tabela 1: Codificação de caracteres.

Char	Binário
b	111
o	00
m	01
e	100
s	101
espaço	110

Fonte: Material didático utilizado pela professora Christine Vieira Scarpato.

Usando essa tabela de codificação, a string "bom esse bombom" ficaria assim:

111 00 01 110 100 101 100 110 111 00 01 111 00 01

Desta forma usamos 39 bits para codificar a string "bom esse bombom", ao invés de 48 bits como mostramos usando a codificação de 3 bits.

De acordo com as observações de MULLER (2000,89p).

> “A probabilidade final da árvore é sempre 1.0, uma vez que necessariamente deve-se atingir 100% das ocorrências de caracteres, permitindo uma codificação total. Uma vez terminada a árvore, basta à formalização da codificação, que é feita com a leitura dos valores binários, da raiz para as folhas. Os valores binários lidos serão o código do percurso da raiz até a folha correspondente ao caractere que se deseja o código.”

MULLER (2000), ainda ressalta que o caractere com freqüência maior tem o valor menor. Este é o objetivo da compactação estatística, permitindo a troca do caractere que possui maior ocorrência em apenas um bit, desta forma acontece com os outros caracteres dispostos em ordem crescente em relação a quantidades de bits de conforme a prioridade de cada um. Devemos ainda salientar, como já foi proposto anteriormente à codificação de Huffman tem como característica de permitir a descompactação direta sem permitir que haja confusão dos bits entre eles.

A partir da leitura das palavras códigos que se encontram no cabeçalho e os símbolos reconstruímos a arvore de Huffman, após a reconstrução percorremos toda a sua extensão de acordo com a disposição de zeros e uns presentes na seqüência binária assim realizando a descompactação.

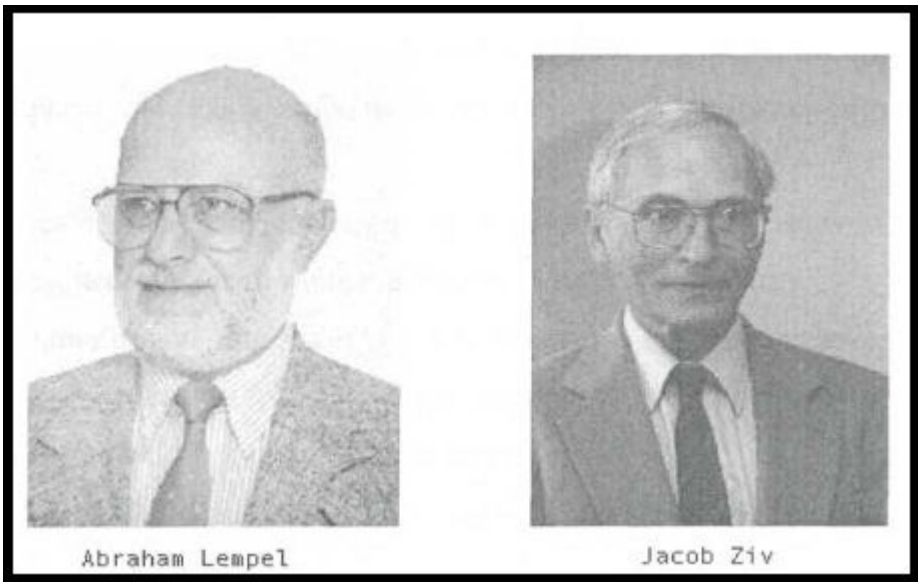
De acordo com FRANÇA. NETO (1998, 134p), "as principais aplicações do algoritmo de Huffman são em dados onde a distribuição de probabilidade não é uniforme". Podemos aplicar o algoritmo de Huffman em varias áreas que necessitem dessas distribuições, também em programas de compactação de arquivos como o PACK/UNPACK existente no Unix e também em muitos outros métodos ou aplicações de compactação como PKZIP da empresa PKWare Incorporation, que de acordo com PROSISE (1993) a técnica reducing do PKZIP aprimora as técnicas utilizadas no algoritmo de Huffman após codificar os dados através de uma técnica de RLE mais complexa.

Durante seus estudos HELD (1992) conclui-o que o algoritmo de Huffman comprime determinado arquivo em 50%, sendo que quanto maior for a redundância dos símbolos, maior será a taxa de compactação. Essa melhora é basicamente obtida utilizando duplas de caracteres e trios caracteres, em vez de caracteres isolados na codificação de Huffman, desta forma comprimiremos seqüências do tipo “ss”, “lh” e “sc” que são duplas comuns dentre de entradas de texto, podendo ser usados em trios de caracteres também.

4 MÉTODO DE LEMPEL-ZIV & WELCH

De acordo com FRANÇA NETO (1998) com o surgimento da compactação baseada em dicionário de dados, também conhecida como cadeia de compactação, a compactação de dados teve uma revolução por todo o mundo, a maioria dos aplicativos de compactação utilizados hoje utilizam algum algoritmo da família LZW criados por Lempel-Ziv & Tery Welch.

Figura 15: Autores do algoritmo LZW.



Fonte: Internet.

Segundo o autor o algoritmo de Lempel-Ziv & Welch é uma variação do algoritmo Lempel-ziv32, este algoritmo foi desenvolvido pelo inglês Tery Welch em 1984, a contribuição de Tery Welch no desenvolvimento do aprimoramento do algoritmo de Lempel-Ziv32 foi tão significativa e importante que existe uma tendência de se desconsiderar o algoritmo original desenvolvido por Tery Welch, por causa da melhora significativa que ele gerou ao algoritmo.

De acordo com WELCH (1984) o aprimoramento do método trabalha da seguinte forma: O algoritmo de Lempel-Ziv32 conhecido com LZ77 utilizava uma janela deslizante para codificar os blocos de caracteres iguais a fim de gerar um código de saída e esse código é o próprio dicionário de strings. O algoritmo trabalha direcionado aos blocos de caracteres mais recentes localizados dentro da janela. Por esse problema Tery Welch no ano seguinte abandonaram o conceito de janela

deslizante e criaram o LZ7833 e começaram a utilizar o dicionário como uma lista ilimitada de blocos, onde os novos blocos, ou novas frases são inseridas em um dicionário e nas próximas vezes que esse bloco aparecer na fonte é gerado um código que as representa no dicionário.

Mesmo Terry Welch sendo o criador do método de compactação LZW ele não é o proprietário da patente, de acordo com a internet espanhola, na época que ele desenvolveu o algoritmo ele trabalhava como pesquisador na Sperry, uma empresa americana que adquiriu os direitos do algoritmo de LZW em 1981, a empresa patenteou o algoritmo, a empresa se fundiu com a Burroughs posteriormente. Atualmente a dona da patente é a UNISYS e ela delimita que todos precisam comprar uma licença para utilizar esse método em aplicações comerciais.

4.1 IDÉIA BÁSICA

FRANÇA NETO (1998, 153), em sua tese explica o método da seguinte forma:

> "A idéia básica deste algoritmo é reaproveitar trechos da seqüência que já foi codificada, usando um sistema de tabela onde essas informações vão sendo armazenadas. Desta forma quanto mais trechos se repetem maior será a taxa de compactação. O algoritmo de Lempel-Ziv-Welch é um método de compactação dinâmico. Ou seja, sua tabela de codificação é formada à medida que os dados vão sendo lidos. Inicialmente, tem-se uma tabela onde está guardada a codificação dos símbolos tomados como básicos. Estes símbolos serão utilizados na formação dos outros elementos da tabela, também chamada de dicionário."

Segundo afirmações de WELCH (1994) seu método LZW consiste em um algoritmo de compactação que tem a capacidade de trabalhar sobre qualquer tipo de entrada de dados, geralmente é um algoritmo rápido tanto na compactação quanto na descompactação das informações sem utilizar operações de ponto flutuante, ou seja, não consome muita memória da máquina e não necessita de maior espaço de armazenamento. O autor considera seu algoritmo como um algoritmo de alto desempenho e de baixo custo computacional.

4.2. SÍNTESE DO ALGORITMO

O algoritmo é claramente dividido em duas partes: a parte onde é feita a compactação da informação e a parte do algoritmo que faz a descompactação da informação. Estudaremos as duas partes separadamente seguindo as definições dos autores Welch (1984) e de Nelson (1989), iniciando pela compactação.

4.2.1 Compactação

Seguiremos as explicações de Mark Nelson (1989) que possui uma definição bem clara do funcionamento do algoritmo de compactação de LZW. De acordo com NELSON (1989) o algoritmo LZW busca sempre gerar códigos para blocos de caracteres (strings) que já são conhecidas, desta forma toda vez que um novo código é gerado, uma nova string é inserida no dicionário de dados, que podemos definir na verdade como uma tabela de strings.

Uma das representações mais clara do algoritmo de compactação LZW é demonstrada na figura 16.

Figura 16: Algoritmo de compactação LZW

```
Carregar o dicionário com strings de tamanho 1;
STRING = caractere da fonte;
ENQUANTO ainda há caracteres FAÇA
        CARACTER = próximo símbolo;
        SE STRING + CARACTER está no dicionário ENTÃO
                STRING = STRING+CARACTER;
        SENÃO
                o código da STRING é passado para a saída do codificador; acrescente
                STRING+CARACTER no dicionário;
                STRING = CARACTER;
        FIM_SE;
FIM_ENQUANTO;
o código da STRING é passado para a saída do codificador;
```

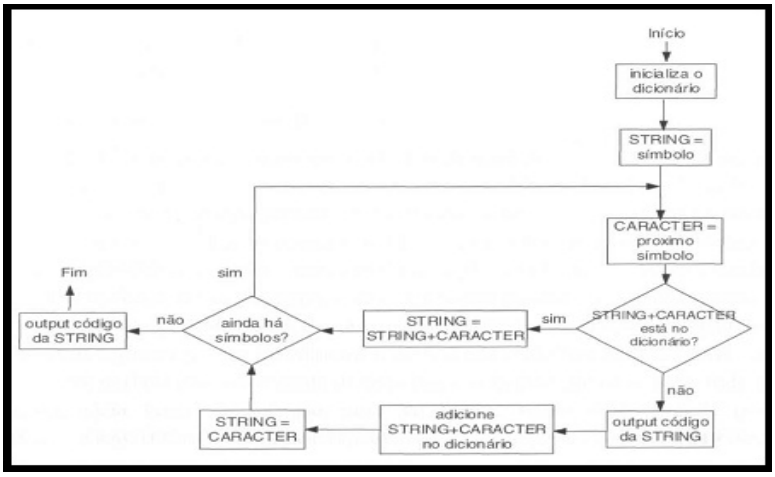
Fonte: adaptado de NELSON (1989).

Segundo NELSON (1989) os blocos de caracteres serão gerados a partir de um prefixo concatenado com um sufixo, na figura 16 que demonstra o algoritmo, podemos dizer que o prefixo é representado pela variável STRING (que também é uma cadeia de caracteres com um ou mais símbolos) e o sufixo é representado pela variável CARACTER (que é um único símbolo).

O algoritmo, figura 16, funciona da seguinte forma: em primeiro lugar, o alfabeto de caracteres é carregado inteiro no dicionário de dados, o alfabeto de caracteres na prática pode ser a própria tabela ASCII, a variável STRING deve receber o primeiro caractere da fonte de dados que conhecemos como prefixo. Em seguida ocorre um processo de repetições conhecido como loop, esse processo é executado enquanto os caracteres da fonte de informação não se esgotam, dentro dessa estrutura de repetição a é atribuída a variável CARACTER o próximo símbolo da fonte que é conhecido como sufixo.

Após isso uma nova cadeia de caracteres é formada, essa nova cadeia é formada por STRING+CARACTER, ou seja, prefixo somado com o sufixo, o algoritmo verifica se a nova cadeia de caracteres está no dicionário; se a cadeia for encontrada no dicionário de dados a STRING terá seu valor concatenado com valor da variável CARACTER, esta por sua vez passará a ser prefixo para os símbolos e a estrutura de repetição é executada novamente. Quando o valor de STRING+CARACTER não for encontrado no dicionário, o código que representa a STRING no dicionário é enviado para a saída do codificador, o valor de STRING+CARACTER é adicionado ao dicionário de dados, a variável STRING recebe a atribuição de valor do CARACTER e a estrutura de repetição é executada novamente. A codificação pode ser representada pelo fluxograma da figura 17.

Figura 17: Fluxograma da compressao LZW



Fonte: adaptado de NELSON (1989).

A partir das definições de NELSON (1989) iremos demonstrar um exemplo de compactação utilizando a seguinte cadeia de caracteres: "ABABC", o dicionário de dados neste caso será iniciado com os três caracteres que compõem o alfabeto, representada por: A, B e C conforme a tabela 2.

Tabela 2: Processo de compactação LZW. Cadeia de entrada ABABC.

DICIONÁRIO				
1	A			
B				
C	STRING	CARACTER		CÓDIGO SAÍDA
4	AB	A	B	1
5	BA	B	A	2
6	ABC	A	B	4
7	-	AB	C	3
8	-	C	-	

Fonte: adaptado de NELSON (1989).

Como podemos observar na primeira passagem pela estrutura de repetição a variável STRING está com o valor do caractere ‘A’ e a variável CARACTER está com o valor de ‘B’. Com a soma de STRING+CARACTER teremos

a seqüência ‘AB’ e essa não está presente no dicionário ainda. A seqüência representa o valor da variável STRING no dicionário na posição 1 da tabela, é gerado como saída no compressor, ‘AB’ que será acrescentada na posição 4 no dicionário de dados e a variável STRING passará a possuir o mesmo valor atribuído a variável CARACTER, ou seja, o valor ‘B’.

Na segunda vez que passa pela estrutura de repetição é adicionada ao valor ‘BA’ à variável STRING e é gerada como saída na posição 2 da tabela que representa a letra ‘B’, e o novo valor da variável STRING é ‘A’

Na terceira passagem pela estrutura de repetição a cadeia ‘AB’ já é encontrada no dicionário de dados na posição 4, desta maneira a variável STRING passa a ser ‘AB’ e o CARACTER receberá o próximo símbolo na entrada, neste caso o ‘C’, neste momento não será produzida nenhuma saída.

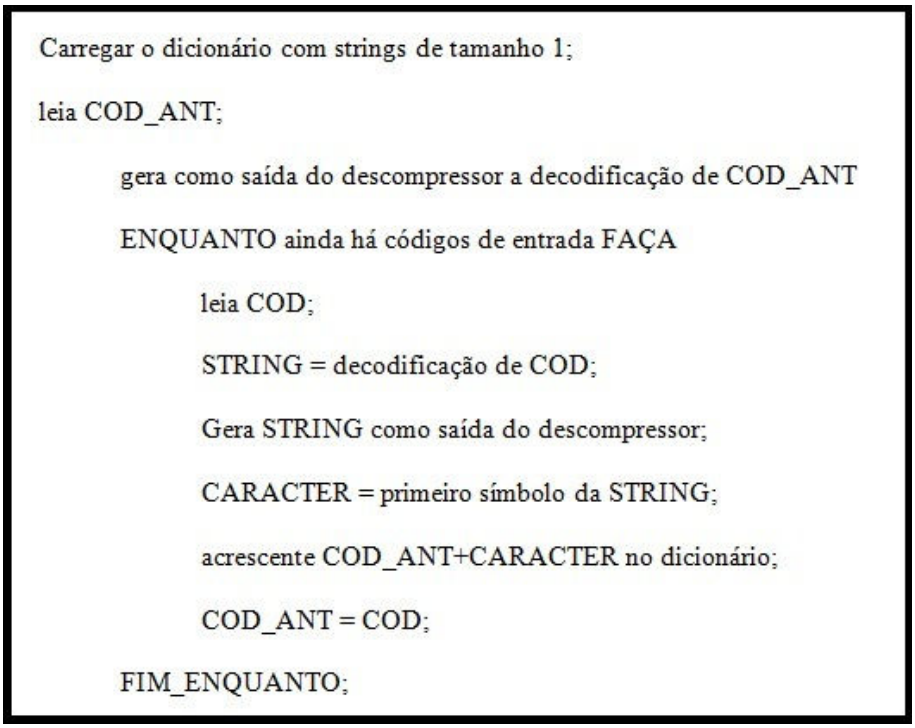
A seguir na próxima vez que a estrutura de dados é executada a cadeia é ‘ABC’ formada por STRING+CARACTER, esta não se encontra no dicionário de dados e a mesma é inserida na posição 6, o código da STRING ‘AB’ é gerada como saída, a STRING recebe o valor de ‘C’ do CARACTER e a estrutura de repetição termina, pois não existem mais símbolos na entrada de dados. No último passo código 3 da STRING é gerado como saída do algoritmo de compactação.

A partir disso a entrada de dados composta da string ‘ABABC’ foi comprimida na seqüência de códigos do dicionário de dados formado por 1 2 4 3. Note que foi gerada uma saída pelo compressor LZW e essa saída é somente uma seqüência de índices pertencentes ao dicionário de dados, e esse representa as cadeias de caracteres com repetições presentes na entrada. Não é necessário enviar o dicionário de dados junto com o código da compactação.

4.2.2 Descompactação

Segundo NELSON (1989) no processo de decodificação, o algoritmo de descompactação tem como entrada o código compactado, durante seu processo e ele vai reconstruir a tabela do dicionário e gerar em sua saída às cadeias de caracteres conforme as originais, que foram codificadas durante o processo de compactação. Podemos escrever o algoritmo de descompactação de LZW conforme a figura 18, considerada umas das mais simples:

Figura 18: Algoritmo de Descompactação LZW.



```
Carregar o dicionário com strings de tamanho 1;
leia COD_ANT;
        gera como saída do descompressor a decodificação de COD_ANT
        ENQUANTO ainda há códigos de entrada FAÇA
                leia COD;
                STRING = decodificação de COD;
                Gera STRING como saída do descompressor;
                CARACTER = primeiro símbolo da STRING;
                acrescente COD_ANT+CARACTER no dicionário;
                COD_ANT = COD;
        FIM_ENQUANTO;
```

Fonte: adaptado de NELSON (1989).

Uma das razões para o algoritmo de LZW ser eficiente é que sua codificação não exige que a tabela de string seja passada junto com o código resultante da compactação, pois durante o processo de compactação o dicionário é totalmente reconstruído de acordo como era durante o processo de compactação.

Mostraremos melhor o processo de descompactação utilizando a tabela 3 sugerida por NELSON (1989), para a saída apresentada pelo compressor. Na tabela de cadeias de símbolos também conhecida como dicionário de dados, a estrutura será idêntica ao dicionário de dados criada na compactação, as primeiras posições da tabela 3 estarão ocupadas por símbolos do alfabeto.

Tabela 3: Processo de descompactação LZW, código de entrada 1 2 4 3.

DICIONÁRIO						
1	A					
2	B					
3	C	STRING	CARACTER	COD_ANT	COD	CÓDIGO SAÍDA
4	AB	B	B	1	2	A
5	BA	AB	A	2	4	B
6	ABC	C	C	4	3	AB
7	-	-	-	3		C

Fonte: adaptado de NELSON (1989).

Na tabela 3 podemos identificar que a variável COD_ANT tem o primeiro código atribuído, o código 1 e por sua vez gera como saída a cadeia de número COD_ANT presente no dicionário de dados, ou seja, na descompactação de COD_ANT=1 a saída será o caractere “A”. Neste momento o algoritmo executa uma estrutura de repetição que é efetuada quando existem códigos de entrada. Dentro desse laço de repetição a variável COD recebe o próximo elemento do código que é 2, por sua vez a variável STRING é atribuída com a descompactação de COD e gera como caractere de saída “B”. A variável CARACTER receberá o primeiro símbolo da STRING “B”. A soma de COD_ANT+CARACTER resulta em “AB” que é adicionado ao dicionário de dados. COD_ANT terá o valor de COD “B”, após isso é executada novamente a estrutura de repetição e a variável COD recebe o valor 4. A variável STRING recebe “AB” que por sua vez é gerada como saída do descompressor. A variável CARACTER recebe o primeiro símbolo pertencente a STRING, que é a letra “A”. COD_ANT somado com CARACTER resulta em “BA” que é adicionado ao dicionário de dados. COD_ANT recebe o valor de COD que é 4. COD recebe o valor 3. A variável STRING passa a ter valor de “C”. A soma das variáveis COD_ANT e CARACTER resulta em “ABC” e este é adicionado ao dicionário. Após isso o algoritmo é encerrado, pois não existem mais códigos a serem descomprimidos na entrada do decodificador.

De acordo com as conclusões de NELSON (1989) existe um determinado código gerado a partir da codificação de compactação do algoritmo de LZW que se torna indecifrável na hora de executar a descompactação, isso ocorre quando existem seqüências de símbolos que consistem em pares STRING e CARACTER

que já foram definidos na tabela de compactação, e o fluxo de entrada de dados é uma seqüência de dados formada por STRING, CARACTER, STRING, CARACTER, STRING. O algoritmo de compactação impossível de ser interpretado pelo descompressor.

Podemos demonstramos esta situação vamos utilizar como exemplo a cadeia de caracteres de entrada “ABABABAB”. Nessa cadeia o símbolo “A” será atribuído a variável STRING, o símbolo “B” será atribuída à variável CARACTER e a cadeia de caracteres “AB” será o par STRING, CARACTER, ou seja, STRING concatenado com CARACTER e teremos a saída do algoritmo de compactação “12352”. Demonstraremos melhor a descompactação através da tabela 4.

Tabela 4: Problema na descompactcao LZW. Código de entrada 1 2 3 5 2.

DICIONÁRIO						
1	A					
2	B	STRING	CARACTER	COD_ANT	COD	CÓDIGO SAÍDA
3	AB	B	B	1	2	A
4	BA	AB	A	2	3	B
5	?	?		3	5	AB

Fonte: adaptado de NELSON (1989).

Podemos observar que o algoritmo de descompactação vai lendo os símbolos de entrada do dicionário de dados e vai gerando como saída à cadeia de caracteres idêntica a original e quando a variável COD recebe 5, a variável STRING receberá a decodificação de COD neste caso isso não pode ser executado porque na posição 5 que será atribuída a COD no dicionário de dados não possui valor definido conforme a tabela 4.

Porém como apenas neste caso não podemos descomprimir o arquivo de entrada por causa dos símbolos indefinidos, podemos modificar e acrescentar uma alteração no algoritmo original. Desta forma teremos o algoritmo alterado conforme a figura 19, este verificará a existência de algum código que não possa ser definido e faz um tratamento para que não ocorram erros neste caso. Na tabela 4 vemos que a rotina de descompactação pega o código 5 que é indefinido no dicionário de dados e após ele decodifica na STRING e o valor da variável COD_ANT que tem valor 3 “AB”

e adiciona o valor de CARACTER que é “A” na STRING. Desta maneira a variável STRING fica com a cadeia “ABA” e executa a tradução correta para o código 5 da sequência expressa por ABABA (STRING + CARACTER + STRING + CARACTER + STRING).

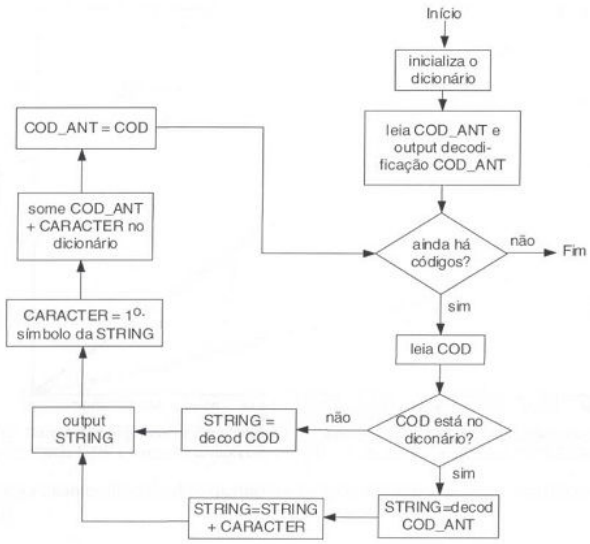
Figura 19: Algoritmo de descompactação LZW com modificações para tratar o problema

```
leia COD_ANT;
gera como saída do descompressor a decodificação de COD_ANT
ENQUANTO ainda há códigos de entrada FAÇA
        leia COD;
        SE COD não está no dicionário ENTÃO
                STRING = decodificação de COD_ANT;
                STRING = STRING+CARACTER;
        SENÃO
                STRING = decodificação de COD;
        FIM_SE;
        Gera STRING como saída do descompressor; CARACTER = primeiro símbolo
        da STRING; acrescente
        COD_ANT+CARACTER no dicionário; COD_ANT =COD;
FIM_ENQUANTO;
```

Fonte: adaptado de NELSON (1989).

Na figura 20 está representado o processo de descompactação já modificado.

Figura 20: Processo de descompactação modificado.



Fonte: adaptado de NELSON (1989).

4.3 APLICAÇÃO DO ALGORITMO

De acordo com FRANÇA NETO (1998, 122p):

> “Uma das aplicações, é a variação implementada no formato GIF, utilizando no armazenamento de imagens. Atua/mente, este é um dos formatos de armazenamento de imagens sem perdas que oferece as melhores taxas de compactação. Também encontrado em outros formatos gráficos”

Podemos citar outras aplicações do algoritmo de LZW, segundo o autor PELISSON (2001), é a recomendação V.42 bis da CCIT (Consultative Committee for International Telephone and Telegraph) e agora denominada ITU (International Telecommunications Union), que utiliza o algoritmo de compactação LZW como padrão para compactação de dados a serem transmitidos nos modems.

De acordo com PELISSON (2001), o método de LZW traz maiores taxas de compactação à medida que o tamanho da entrada vai crescendo. Quando maior for à entrada a ser codificada, maior o dicionário e menor a saída. Esta questão traz vantagens e desvantagens ao algoritmo. A vantagem está em trazer resultados significativos quando lida com arquivos de grande porte. A desvantagem, ao lidar com esses tipos de arquivos, está no tamanho do dicionário. Computacionalmente, o armazenamento do dicionário formado durante a codificação e a constante

verificação de seus componentes pode ser muito caro. Outro problema é que a compactação LZW às vezes pode formar muitas cadeias com poucos símbolos no dicionário. Este problema pode ser solucionado com uma variação da técnica LZW chamado Algoritmo de LZMW, que executa uma formação de novas cadeias mais longas e de forma mais rápida.

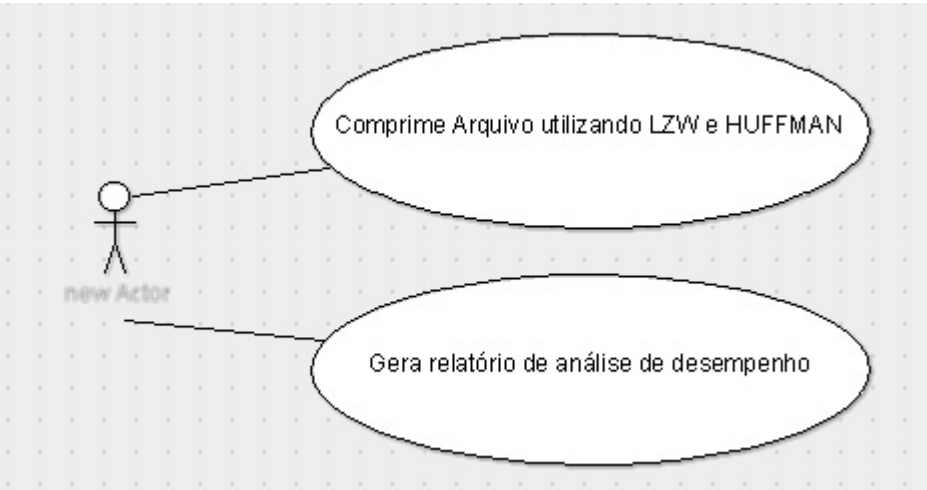
5 DESENVOLVIMENTO DA FERRAMENTA

Modelamos o sistema proposto de acordo com as orientações da UML, desenvolvemos a modelagem utilizando a ferramenta Argo UML. Utilizamos na modelagem da ferramenta os diagramas de use-case e diagrama de seqüência. Nesta ferramenta teremos duas partes distintas onde teremos a interface que será vista pelo usuário e depois teremos o núcleo interno que conterá todas as funções de compactação de arquivos. O núcleo interno que possui as funções é executado terá a função de buscar os arquivos a serem comprimidos e por fim comprimi-los, utilizando os métodos de Huffman e Lempel e Ziv, esses algoritmos são utilizados por uma vasta quantidade de aplicativos similares disponíveis na internet, essa compactação gerada no final resulta em um arquivo binário que ao final pode ser reconstruído na sua originalidade total.

5.1 DIAGRAMAS DE CASO DE USO (USE CASE)

Nesta ferramenta teremos dois casos de uso, o primeiro caso de uso se trata da operação de compactação do arquivo e no segundo a relação de resultados apresentados após a compactação.

Figura 21: Caso de uso do sistema.

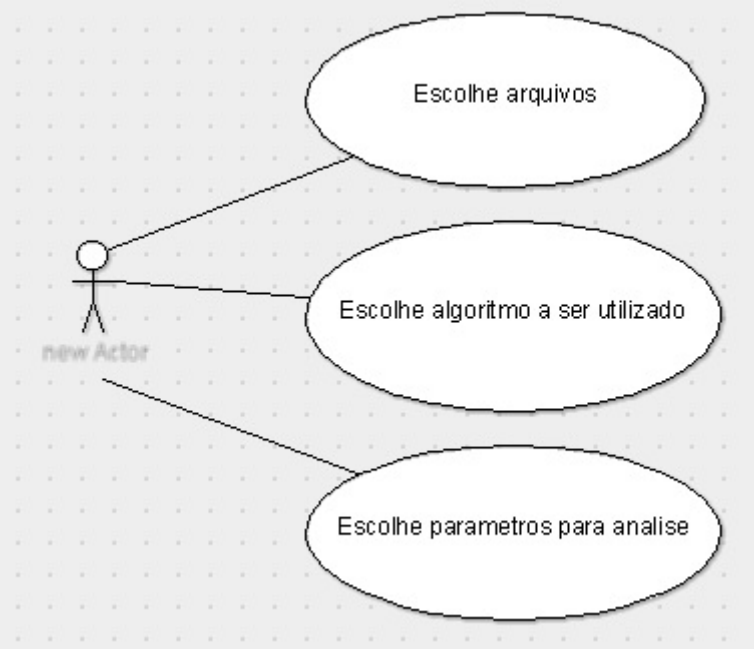


Fonte: do autor.

Durante a compactação do arquivo teremos as opções demonstradas através da figura 22.

1. Escolhe arquivos: O usuário deve escolher o arquivo para a compactação que será feita através do núcleo funcional.
2. Escolhe algoritmo: O usuário deve escolher qual algoritmo deseja analisar.
3. Escolhe os parâmetros para análise: O usuário define os parâmetros a serem utilizados na comparação dos algoritmos, como o que analisar, e se deseja comparar os resultados.

Figura 22: Caso de uso do sistema, escolha de arquivos.



Fonte: do autor.

5.2 DIAGRAMA DE SEQÜÊNCIA

A comunicação entre a Interface do sistema e o núcleo funcional é feita através de mensagens enviadas de um processo para o outro, durante este procedimento o sistema obedecera a seguinte seqüência:

1. O usuário selecionara os arquivos a serem comprimidos e os parâmetros para a análise deles, e se existira comparação entre os arquivos.

2. A interface do sistema enviará uma mensagem para o núcleo funcional, essa mensagem será o arquivo escolhido pelo usuário e os parâmetros selecionados.

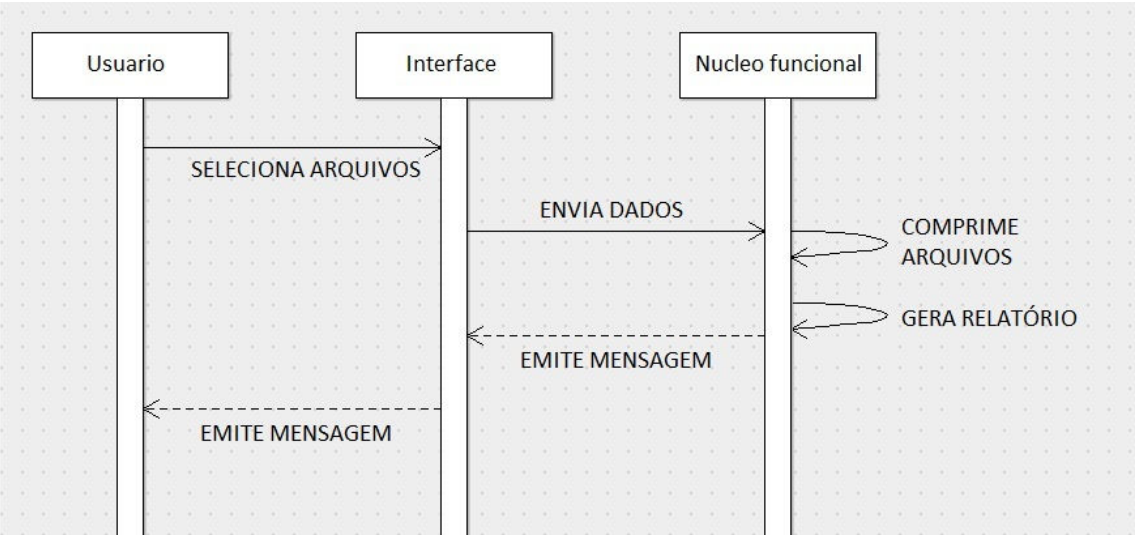
3. O núcleo funcional receberá mensagem com as informações e ira realizar o processo de compactação e análise de desempenho, se for selecionado o parâmetro de comparação, também ira executá-la.

4. Depois de comprimir os arquivos, analisar o desempenho e comparar os arquivos o núcleo funcional retorna uma mensagem a interface informando que o processo foi concluído.

5. Nisso a interface será responsável para informar o usuário que o processo foi executado corretamente e emite o relatório com a análise do desempenho e se foi selecionado a comparação entre algoritmos ele emite os resultados dessa comparação.

A figura 23 exemplifica melhor o funcionamento através de um diagrama de seqüência.

Figura 23: Diagrama de seqüência do sistema.



Fonte: do autor.

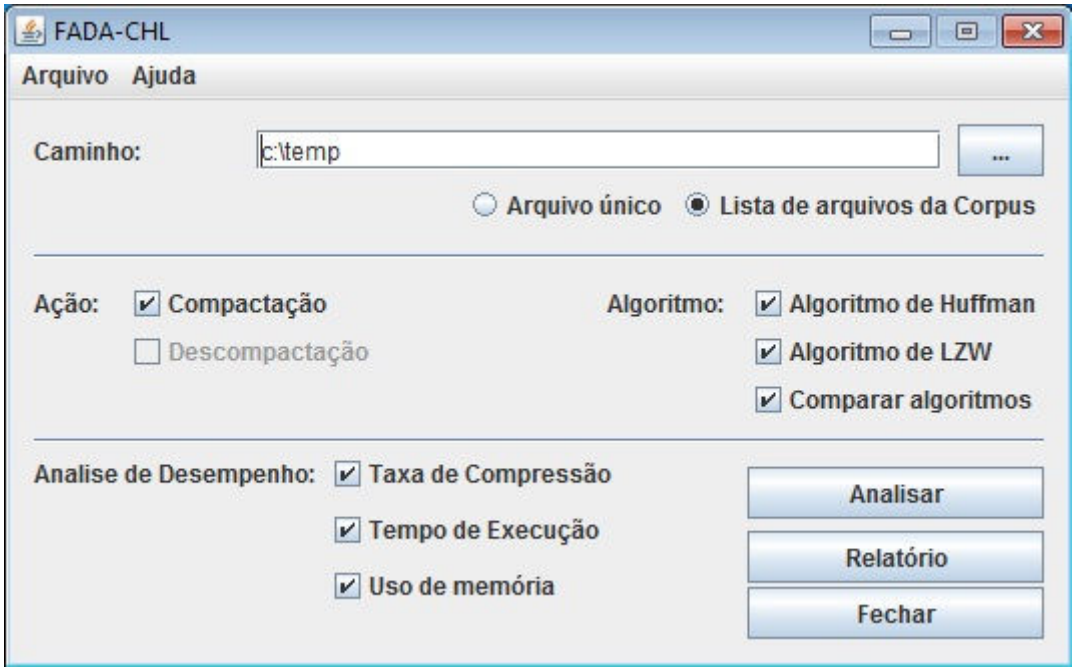
O sistema de compactação será executado da seguinte forma:

- Arquivos da entrada serão analisados;
- O núcleo funcional executara a compactação de acordo com os parâmetros escolhidos.
- O núcleo aplicara os métodos de análise de algoritmo.
- Serão retornados a partir do núcleo os resultados sobre a análise e comparação.

5.3 TELAS DO SISTEMA

Nosso sistema foi dividido em duas partes, onde na primeira parte temos a Interface e na segunda o núcleo funcional, o núcleo funcional será a parte interna do sistema que possuirá as funções de compactação e análise de desempenho, esse núcleo não possui telas, a Interface possuirá quatro telas. O usuário vai interagir com o sistema através da Interface, nela selecionara os arquivos a serem comprimidos pelo compressor, selecionara as opções de análise de desempenho, selecionara quais os algoritmos a serem utilizados, selecionara se ira compará-los, todas as operações a serem solicitadas na interface serão realizadas pelo núcleo principal.

Figura 24: Tela inicial do sistema.

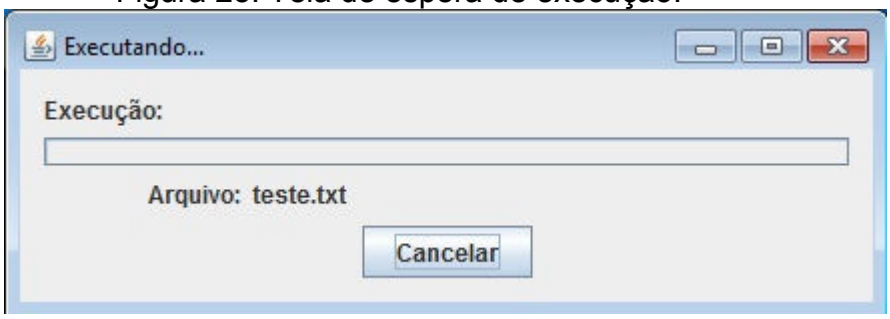


Fonte: do autor.

Na figura 24 é mostrada a janela principal do sistema, onde encontramos as opções mencionadas anteriormente, parâmetros para análise de desempenho, abertura de arquivos e emissão do relatório de comparação.

- Caminho: Neste campo selecionamos o diretório ou arquivo a ser compactado.
- Ação: Neste campo selecionamos a ação a ser executada, neste caso só a compactação estará disponível.
- Algoritmo: Neste campo selecionamos os algoritmos a ser utilizado.
- Análise de Desempenho: Neste campo selecionamos o que analisar.

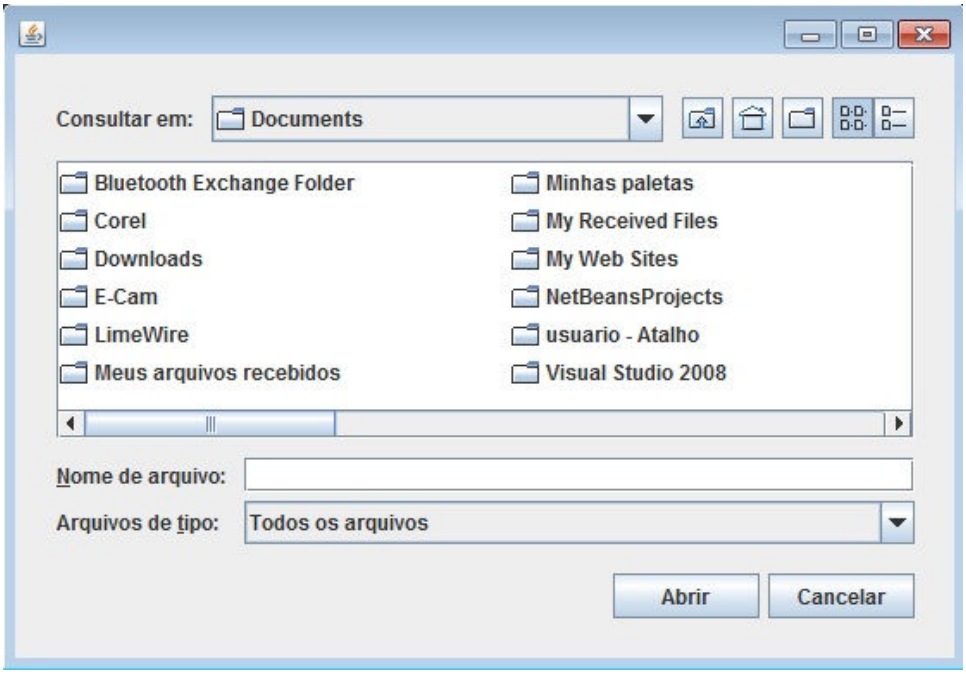
Figura 25: Tela de espera de execução.



Fonte: do autor.

Na figura 25 é mostrada a janela de espera de execução, durante o processo de compactação, para demonstrar para usuário que o processo esta em execução quando são selecionados muitos arquivos.

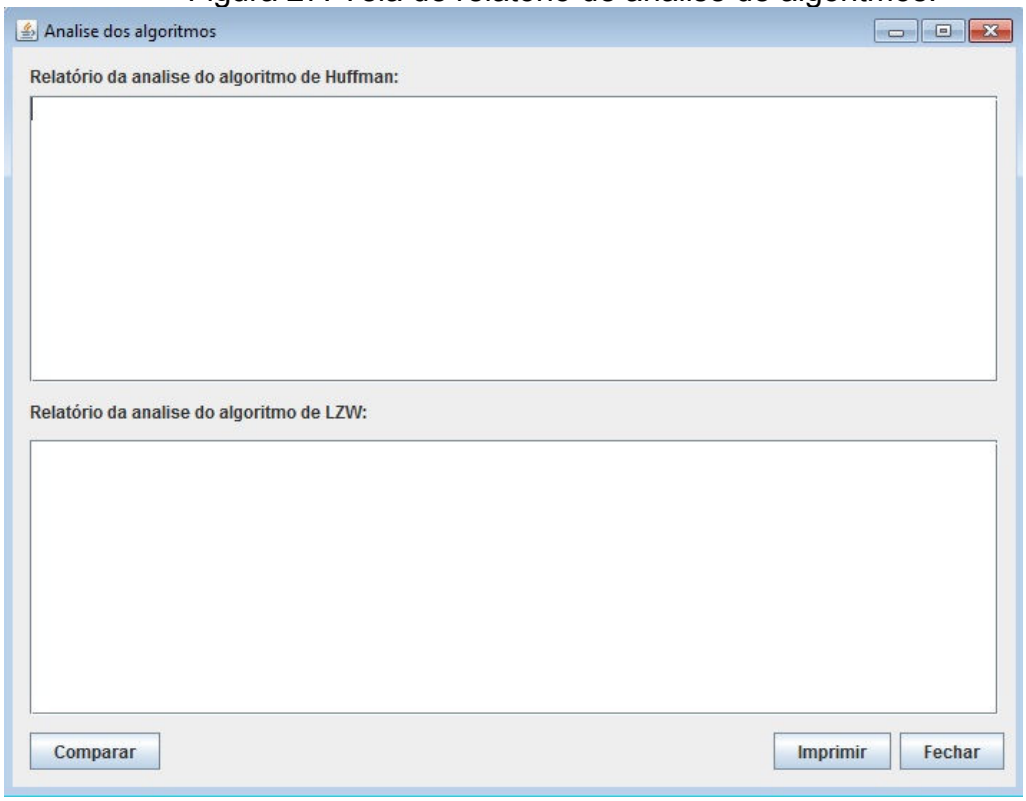
Figura 26: Tela de abertura de arquivos.



Fonte: do autor.

Na figura 26 temos a janela de abertura de arquivos, onde selecionamos os arquivos a serem compactados.

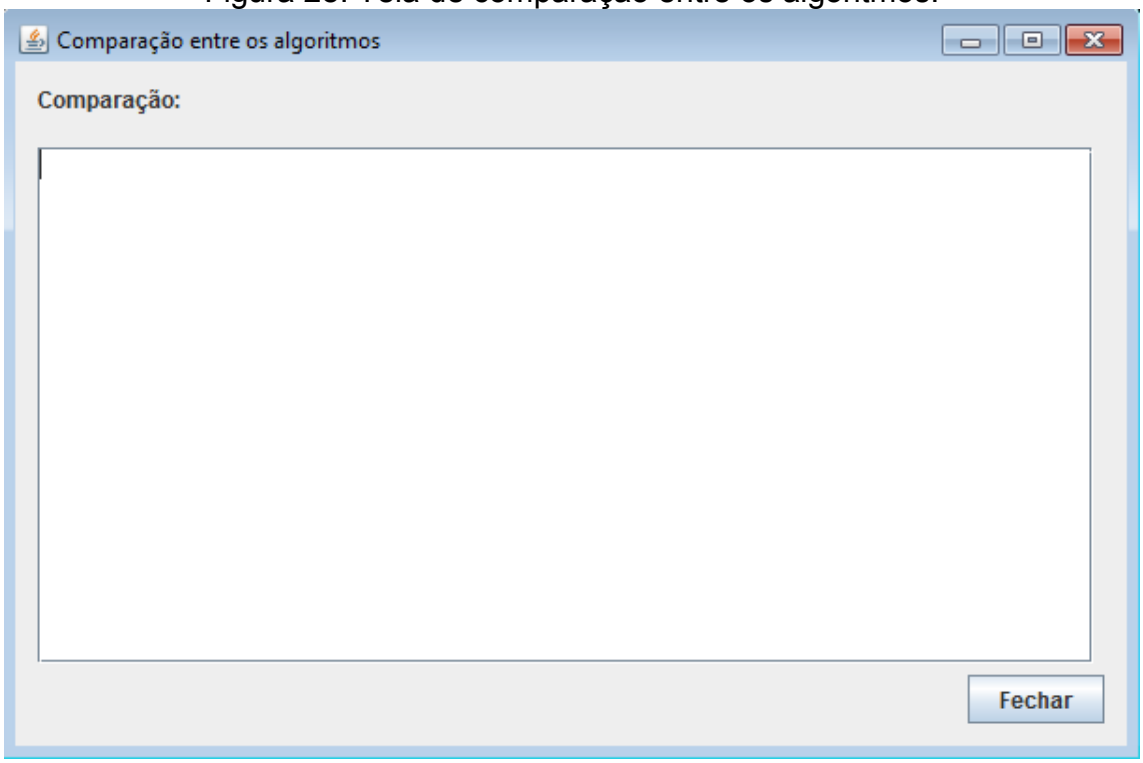
Figura 27: Tela de relatório de análise de algoritmos.



Fonte: do autor.

Na figura 27 temos a janela de análise dos algoritmos onde são apresentados os resultados de referentes a análise do algoritmo de Huffman e do algoritmo de lZW. Inicialmente esses resultados serão apresentados através de texto, podendo ser atualizado em uma versão futura para gráficos e tabela.

Figura 28: Tela de comparação entre os algoritmos.



Fonte: do autor.

Na figura 28 temos a janela de comparação dos resultados dos algoritmos onde são apresentados os resultados de referentes à comparação entre o algoritmo de Huffman e o algoritmo de lZW. Inicialmente esses resultados serão apresentados através de texto, podendo ser atualizado em uma versão futura para gráficos e tabela.

5.4 LINGUAGUEM DESENVOLVIDA: JAVA

De acordo com MENGUE (2002), em 1991 um grupo de engenheiros da Sun Microsystems foi encarregado de criar uma nova linguagem que pudesse ser utilizada em pequenos equipamentos como controles de TV, telefones, fornos,

geladeiras, etc. Essa linguagem deveria dar a esses aparelhos a capacidade de se comunicar entre si, para que a casa se comportasse como uma federação. Deveria ainda ser capaz de gerar códigos muito pequenos, que pudessem ser executados em vários aparelhos diferentes, e praticamente infalível. Os engenheiros escolheram o C++ como ponto de partida. Orientada a objetos, muito poderosa e gerando pequenos programas, parecia a escolha correta. Para solucionar o problema da execução em várias arquiteturas, eles utilizaram o conceito da máquina virtual, onde cada fabricante iria suportar algumas funções básicas que os programas utilizariam. Até hoje a linguagem resultante deste projeto não é utilizada em aparelhos eletrodomésticos. Ao invés disso, o Java se tornou um das linguagens de programação mais utilizadas no planeta.

5.4.1 VANTAGENS NA UTILIZAÇÃO DO JAVA

Na maioria das linguagens de programação, você precisa compilar ou interpretar um programa para que ele seja executado em seu computador. A linguagem Java é diferente, pois seus programas são compilados **E** interpretados. Com o compilador, você inicialmente transforma seu programa em uma linguagem intermediária, chamada *bytecode*. Esse código é independente de plataforma, e é mais tarde interpretado por um interpretador Java. A compilação acontece apenas uma vez; a interpretação acontece todas as vezes que seu programa é executado.

5.4.2 PLATAFORMA JAVA

Por plataforma, entendemos o conjunto de hardware e software no qual um programa executa. Alguns exemplos de plataformas muito usadas são o Windows, o Linux, o MacOS. A plataforma Java é diferente, pois não envolve hardware; ela utiliza a plataforma de hardware das outras.

A plataforma Java tem dois componentes:

- Java Virtual Machine (Java VM ou JVM)
- Java Application Programming Interface (Java API)

A API Java é uma coleção de componentes de software prontos, que incluem desde estruturas para manipulação de arquivos até a construção de aplicativos gráficos. A API é organizada como um grupo de bibliotecas com classes e interfaces; essas bibliotecas são chamadas de pacotes.

6 ANÁLISE DE DESEMPENHO DOS ALGORITMOS HUFFMAN E LZW

Neste projeto iremos executar a comparação dos métodos de Huffman e LZW utilizando a ferramenta desenvolvida FADA-CHL, compararemos os algoritmos entre eles mesmos e realizaremos uma análise dos resultados, compararemos também esses resultados com os de uma ferramenta de compactação. Esta comparação tem como base analisar certos parâmetros de desempenho como a taxa de compactação, tempo de execução e taxa de uso da memória.

Taxa de compactação – A taxa de compactação será sempre demonstrada em bits por byte, essa taxa indica quantos bits são necessários em média para representar cada caractere depois da execução do algoritmo de compactação.

Tempo de Execução – O tempo de execução é o tempo necessário gasto para que o método de compactação ou o método de descompactação seja executado e apresente seus resultados.

Taxa de uso da Memória – A taxa de uso da memória demonstra computacionalmente a quantidade de memória gasto por cada método na execução tanto de compactação, quanto de descompactação.

Para as comparações que faremos também incluiremos um software de compactação conhecido e muito utilizado, o WINRAR na versão 3.93, conseguimos este software em sua versão livre na internet facilmente. Incluiremos esse software para os testes para formarmos uma ponte para os dois tipos de comparação que executaremos, lembrando que elas foram desenvolvidas em diferentes ambientes.

Utilizaremos como benchmark o CanterBury Corpus para a comparação, pois é a principal coleção de arquivos utilizados para a comparação de algoritmos de compactação, essa coleção esta disponível na pagina na internet www.corpus.canterbury.ac.nz, os arquivos presentes nessa coleção possuem uma característica de possuírem resultados típicos atualmente nos algoritmos de compactação.

O Corpus Canterbury é uma referência para permitir que pesquisadores para avaliar métodos de compactação sem perdas. Corpus Canterbury, um substituto para o Corpus Calgary. O conjunto de arquivos Canterbury Corpus vem sendo desenvolvido especificamente para testar novos algoritmos de compactação. Os arquivos foram selecionados com base em sua capacidade de fornecer resultados de desempenho representativos.

Este conjunto de arquivos foi projetado para substituir o Corpus Calgary, que agora é mais de dez anos de idade. O Corpus Calgary foi apresentado pela primeira vez no livro de compactação de texto por Bell, Cleary e Witten, publicado em 1990. Resultados de arquivos a partir do corpus foram relatados por vários pesquisadores desde então e vários conjuntos de resultados estão disponíveis neste site www.corpus.canterbury.ac.nz.

Tabela 5. Relação de arquivos do Canterbury Corpus.

Número	**Nome arquivo**	**Tamanho(em bytes)**
1	Alice29.txt	152.089
2	Asyoulik.txt	125.179
3	Cp.html	24.603
4	Fields.c	11.150
5	Grammar.lsp	3.721
6	Kennedy.xls	1.029.744
7	Lcet10.txt	426.754
8	Plrabn.txt	481.861
9	Ptt5	513.216
10	Sum	38.240
11	Xagrs.1	4.227

Fonte: do autor.

Realizamos os testes em um computador no padrão IBM-PC com processador Intel Atom 1.6Ghz e 2Gb de memória e sistema operacional de Windows Seven. Apresentaremos os resultados das comparações através dos

relatórios gerados a partir do sistema desenvolvido. Para efetuarmos os testes dividiremos as etapas em três, de acordo com os subtítulos 6.1, 6.2 e 6.3 onde abordaremos as taxas de compactação, tempo de execução e memória utilizada. Escolheremos 9 arquivos da tabela 5 e realizaremos os testes através da ferramenta FADA-CHL, para um resultado mais correto executaremos a análise de desempenho 5 vezes e através disso geraremos uma média para o tempos de execução, taxas de compactação e memória utilizada.

6.1 TAXA DE COMPACTAÇÃO

Como primeiro resultado da análise de desempenho de nosso sistema teremos a taxa de compactação que pode ser expressa em percentual, essa taxa expressa o percentual de redução do arquivo de origem em relação ao arquivo de destino. Escolhemos utilizar essa medida por ser uma forma clara e direta de comparação entre os arquivos gerados pelos algoritmos de compactação.

6.1.1 Arquivo 1: Alice29.txt

Escolhemos o arquivo Alice29.txt que tem 152.089 bytes de informação, na tabela 6 são apresentados os resultados obtidos para taxa de compactação através da FADA-CHL.

Tabela 6: Execução da ferramenta FADA-CHL para o arquivo Alice29.txt

N. Execução	HUFFMAN	LZW
1	51,41	57,37
2	51,41	57,37
3	51,41	57,37
4	51,41	57,37
5	51,41	57,37
Média	**51,41**	**57,37**

Fonte: do autor.

Para o arquivo Alice29.txt obtemos como taxa de compactação media com o algoritmo de Huffman o valor de 51,41 e com o algoritmo de LZW o valor de 57,37 em relação ao arquivo original, o algoritmo de HUFFMAN obteve a maior taxa de compactação, pois compactou o arquivo original em um arquivo com tamanho 48,59% menor que o arquivo original, enquanto o algoritmo LZW compactou o arquivo original com tamanho 42,63% menor.

6.1.2 Arquivo 2: Asyoulik.txt

Escolhemos o arquivo Asyoulik.txt que tem 125.179 bytes de informação, na tabela 7 são apresentados os resultados obtidos para taxa de compactação através da FADA-CHL.

Tabela 7: Execução da ferramenta FADA-CHL para o arquivo Asyoulik.txt

N. Execução	HUFFMAN	LZW
1	54,03	61,23
2	54,03	61,23
3	54,03	61,23
4	54,03	61,23
5	54,03	61,23
Média	**54,03**	**61,23**

Fonte: do autor.

Para o arquivo Asyoulik.txt obtemos como taxa de compactação média com o algoritmo de Huffman o valor de 54,03 e com o algoritmo de LZW o valor de 61,23 em relação ao arquivo original, o algoritmo de HUFFMAN obteve a maior taxa de compactação, pois compactou o arquivo original em um arquivo com tamanho 45,97% menor que o arquivo original, enquanto o algoritmo LZW compactou o arquivo original com tamanho 38,77% menor.

6.1.3 Arquivo 3: Cp.html

Escolhemos o arquivo Cp.html que tem 24.603 bytes de informação, na tabela 8 são apresentados os resultados obtidos para taxa de compactação através da FADA-CHL.

Tabela 8: Execução da ferramenta FADA-CHL para o arquivo Cp.html

N. Execução	HUFFMAN	LZW
1	58,15	68,72
2	58,15	68,72
3	58,15	68,72
4	58,15	68,72
5	58,15	68,72
Média	**58,15**	**68,72**

Fonte: do autor.

Para o arquivo Cp.html obtemos como taxa de compactação media com o algoritmo de Huffman o valor de 58,15% e com o algoritmo de LZW o valor de 68,72% em relação ao arquivo original, o algoritmo de HUFFMAN obteve a maior taxa de compactação, pois compactou o arquivo original em um arquivo com tamanho 41,85% menor que o arquivo original, enquanto o algoritmo LZW compactou o arquivo original com tamanho 31,28% menor.

6.1.4 Arquivo 4: Fields.c

Escolhemos o arquivo Fields.c que tem 11.150 bytes de informação, na tabela 9 são apresentados os resultados obtidos para taxa de compactação através da FADA-CHL.

Tabela 9: Execução da ferramenta FADA-CHL para o arquivo Fields.c

N. Execução	HUFFMAN	LZW
1	58,31	73,67
2	58,31	73,67
3	58,31	73,67
4	58,31	73,67
5	58,31	73,67
Média	**58,31**	**73,67**

Fonte: do autor.

Para o arquivo Fields.c obtemos como taxa de compactação media com o algoritmo de Huffman o valor de 58,31% e com o algoritmo de LZW o valor de 73,67% em relação ao arquivo original, o algoritmo de HUFFMAN obteve a maior taxa de compactação, pois compactou o arquivo original em um arquivo com tamanho 41,69% menor que o arquivo original, enquanto o algoritmo LZW compactou o arquivo original com tamanho 26,33% menor.

6.1.5 Arquivo 5: Grammar.lsp

Escolhemos o arquivo Grammar.lsp que tem 3.721 bytes de informação, na tabela 10 são apresentados os resultados obtidos para taxa de compactação através da FADA-CHL.

Tabela 10: Execução da ferramenta FADA-CHL para o arquivo Grammar.lsp

N. Execução	HUFFMAN	LZW
1	58,72	83,39
2	58,72	83,39
3	58,72	83,39
4	58,72	83,39
5	58,72	83,39
Média	**58,72**	**83,39**

Fonte: do autor.

Para o arquivo Grammar.lsp obtemos como taxa de compactação media com o algoritmo de Huffman o valor de 58,72% e com o algoritmo de LZW o valor de 83,39% em relação ao arquivo original, o algoritmo de HUFFMAN obteve a maior taxa de compactação, pois compactou o arquivo original em um arquivo com tamanho 41,28% menor que o arquivo original, enquanto o algoritmo LZW compactou o arquivo original com tamanho 16,61% menor.

6.1.7 Arquivo 7: Lcet10.txt

Escolhemos o arquivo Lcet10.txt que tem 426.754 bytes de informação, na tabela 11 são apresentados os resultados obtidos para taxa de compactação através da FADA-CHL.

Tabela 11: Execução da ferramenta FADA-CHL para o arquivo Lcet10.txt

N. Execução	HUFFMAN	LZW
1	52,25	52,87
2	52,25	52,87
3	52,25	52,87
4	52,25	52,87
5	52,25	52,87
Média	**52,25**	**52,87**

Fonte: do autor.

Para o arquivo Lcet10.txt obtemos como taxa de compactação media com o algoritmo de Huffman o valor de 52,25% e com o algoritmo de LZW o valor de 52,87% em relação ao arquivo original, o algoritmo de HUFFMAN obteve a maior taxa de compactação, pois compactou o arquivo original em um arquivo com tamanho 47,75% menor que o arquivo original, enquanto o algoritmo LZW compactou o arquivo original com tamanho 47,13% menor.

6.1.8 Arquivo 8: Plrabn.txt

Escolhemos o arquivo Plrabn.txt que tem 481.861 bytes de informação, na tabela 12 são apresentados os resultados obtidos para taxa de compactação através da FADA-CHL.

Tabela 12: Execução da ferramenta FADA-CHL para o arquivo Plrabn.txt

N. Execução	HUFFMAN	LZW
1	50,89	56,29
2	50,89	56,29
3	50,89	56,29
4	50,89	56,29
5	50,89	56,29
Média	**50,89**	**56,29**

Fonte: do autor.

Para o arquivo Plrabn.txt obtemos como taxa de compactação media com o algoritmo de Huffman o valor de 50,89% e com o algoritmo de LZW o valor de 56,29% em relação ao arquivo original, o algoritmo de HUFFMAN obteve a maior taxa de compactação, pois compactou o arquivo original em um arquivo com tamanho 49,11% menor que o arquivo original, enquanto o algoritmo LZW compactou o arquivo original com tamanho 43,71% menor.

6.1.9 Arquivo 9: Ptt5

Escolhemos o arquivo Ptt5 que tem 513.216 bytes de informação, na tabela 13 são apresentados os resultados obtidos para taxa de compactação através da FADA-CHL.

Tabela 13: Execução da ferramenta FADA-CHL para o arquivo Ptt5

N. Execução	HUFFMAN	LZW
1	56,75	12,02
2	56,75	12,03
3	56,75	12,03
4	56,75	12,03
5	56,75	12,03
Média	**56,75**	**12,03**

Fonte: do autor.

Para o arquivo Ptt5 obtemos como taxa de compactação media com o algoritmo de Huffman o valor de 56,75% e com o algoritmo de LZW o valor de 12,03% em relação ao arquivo original, o algoritmo de LZW obteve a maior taxa de compactação, pois compactou o arquivo original em um arquivo com tamanho 87,97% menor que o arquivo original, enquanto o algoritmo HUFFMAN compactou o arquivo original com tamanho 43,25% menor.

6.1.11 Arquivo 11: Xagrs.1

Escolhemos o arquivo Xagrs.1 que tem 4.227 bytes de informação, na tabela 14 é apresentado os resultados obtidos para taxa de compactação através da FADA-CHL.

Tabela 14: Execução da ferramenta FADA-CHL para o arquivo Xagrs.1

N. Execução	HUFFMAN	LZW
1	60,77	92,64
2	60,77	92,64
3	60,77	92,64
4	60,77	92,64
5	60,77	92,64
Média	**60,77**	**92,64**

Fonte: do autor.

Para o arquivo Xagrs.1 obtemos como taxa de compactação media com o algoritmo de Huffman o valor de 60,77% e com o algoritmo de LZW o valor de 92,64 em relação ao arquivo original, o algoritmo de LZW obteve a maior taxa de compactação, pois compactou o arquivo original em um arquivo com tamanho 39,23% menor que o arquivo original, enquanto o algoritmo HUFFMAN compactou o arquivo original com tamanho 7,36% menor. Na tabela 15 temos as taxas medias de compactação de cada algoritmo em relação à taxa de compactação do WINRAR para os arquivos da tabela 5.

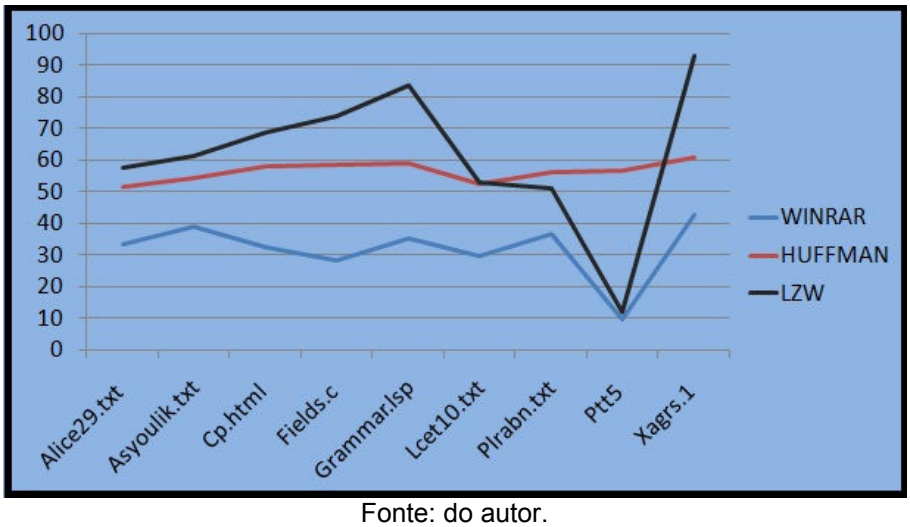
Tabela 15. Taxas de compactação¸
dos arquivos do Canterbury Corpus.

Num. Arq	**WINRAR**	**HUFFMAN**	**LZW**
1	33,70	51,41	57.37
2	39,26	54,03	61,23
3	32,42	58,15	68,72
4	28,28	58,31	73,677
5	35,25	58,72	83,39
6	----------	----------	----------
7	29,75	52,25	52,87
8	36,54	56,29	50,89
9	9,69	56,75	12,03
10	----------	----------	----------
11	42,84	60,77	92,64
Média	**31,97**	**56,29**	**61,42**

Fonte: do autor.

Na tabela 16 temos as taxas médias de compactação dos algoritmos de compactação em relação aos arquivos da tabela 5.

Figura 21. Gráfico de taxas de compactação, dos arquivos do Canterbury Corpus.



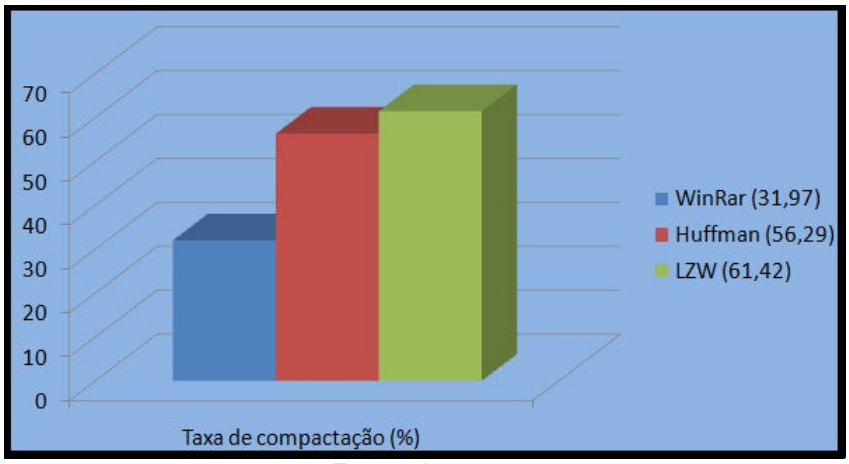
Fonte: do autor.

Tabela 16. Taxas médias de compactação dos métodos.

Método	%
WINRAR	31,97
HUFFMAN	56,29
LZW	61,42

Fonte: do autor.

Figura 22. Gráfico de taxas médias de compactação dos métodos.



Fonte: do autor.

6.2 TEMPO DE EXECUÇÃO

Como segundo resultado da nossa análise de desempenho temos um fator importante que é o tempo de execução, ou seja, o tempo gasto para que o método de compactação seja executado. Para medir essa taxa de execução devemos lembrar que não é tão fácil pois essa taxa sofre influencia da arquitetura computacional utilizada e de se o compilador utilizado é bom em questão a velocidade. Os tipos de arquivos a serem compactados também podem influenciar no tempo de execução, na tabela XX mostraremos os tempos de compactação expressos em milissegundos, para tempos abaixo de 1 milissegundo utilizaremos arredondamento para facilitar o entendimento.

6.2.1 Arquivo 1: Alice29.txt

Escolhemos o arquivo Alice29.txt que tem 152.089 bytes de informação, na tabela 17 são apresentados os resultados obtidos para o tempo de execução em segundos através da FADA-CHL.

Tabela 17: Execução da ferramenta FADA-CHL para o arquivo Alice29.txt

N. Execução	HUFFMAN	LZW
1	6,314	262,604
2	6,306	257,906
3	6,25	262,05
4	6,084	257,618
5	6,203	264,781
Média	**6,23**	**260,99**

Fonte: do autor.

Para o arquivo Alice29.txt obtemos como tempo de execução médio com o algoritmo de Huffman o valor de 6,23 segundos e com o algoritmo de LZW o valor de 260,99 segundos, o algoritmo de HUFFMAN tem o tempo de execução melhor que o algoritmo de LZW.

6.2.2 Arquivo 2: Asyoulik.txt

Escolhemos o arquivo Asyoulik.txt que tem 125.179 bytes de informação, na tabela 18 são apresentados os resultados obtidos para o tempo de execução através da FADA-CHL.

Tabela 18: Execução da ferramenta FADA-CHL para o arquivo Asyoulik.txt

N. Execução	HUFFMAN	LZW
1	5,28	199,435
2	5,348	204,888
3	5,285	210,035
4	5,572	209,021
5	5,391	206,347
Média	**5,39**	**205,94**

Fonte: do autor.

Para o arquivo Asyoulik.txt obtemos como o tempo de execução médio com o algoritmo de Huffman o valor de 5,39 segundos e com o algoritmo de LZW o valor de 205,94 segundos o algoritmo de HUFFMAN tem o tempo de execução melhor que o algoritmo de LZW.

6.2.3 Arquivo 3: Cp.html

Escolhemos o arquivo Cp.html que tem 24.603 bytes de informação, na tabela 19 são apresentados os resultados obtidos para o tempo de execução através da FADA-CHL.

Tabela 19: Execução da ferramenta FADA-CHL para o arquivo Cp.html

N. Execução	HUFFMAN	LZW
1	1,409	13,478
2	1,307	12,205
3	1,474	11,614
4	1,429	12,184
5	1,611	15,953
Média	**1,44**	**13,08**

Fonte: do autor.

Para o arquivo Cp.html obtemos como o tempo de execução médio com o algoritmo de Huffman o valor de 1,44 segundos e com o algoritmo de LZW o valor de 13,08 segundos o algoritmo de HUFFMAN tem o tempo de execução melhor que o algoritmo de LZW.

6.2.4 Arquivo 4: Fields.c

Escolhemos o arquivo Fields.c que tem 11.150 bytes de informação, na tabela 20 são apresentados os resultados obtidos para o tempo de execução através da FADA-CHL.

Tabela 20: Execução da ferramenta FADA-CHL para o arquivo Fields.c

N. Execução	HUFFMAN	LZW
1	0,606	4,707
2	0,638	4,136
3	0,631	4,192
4	0,774	4,258
5	0,631	4,28
Média	**0,65**	**4,31**

Fonte: do autor.

Para o arquivo Fields.c obtemos como o tempo de execução médio com o algoritmo de Huffman o valor de 0,65 segundos e com o algoritmo de LZW o valor de 4,31 segundos o algoritmo de HUFFMAN tem o tempo de execução melhor que o algoritmo de LZW.

6.2.5 Arquivo 5: Grammar.lsp

Escolhemos o arquivo Grammar.lsp que tem 3.721 bytes de informação, na tabela 21 são apresentados os resultados obtidos para o tempo de execução através da FADA-CHL.

Tabela 21: Execução da ferramenta FADA-CHL para o arquivo Grammar.lsp

N. Execução	HUFFMAN	LZW
1	0,277	0,572
2	0,264	0,622
3	0,247	0,545
4	0,251	0,541
5	0,275	0,653
Média	**0,26**	**0,58**

Fonte: do autor.

Para o arquivo Grammar.lsp obtemos como o tempo de execução médio com o algoritmo de Huffman o valor de 0,26 segundos e com o algoritmo de LZW o valor de 0,58 segundos o algoritmo de HUFFMAN tem o tempo de execução melhor que o algoritmo de LZW.

6.2.7 Arquivo 7: Lcet10.txt

Escolhemos o arquivo Lcet10.txt que tem 426.754 bytes de informação, na tabela 22 são apresentados os resultados obtidos para o tempo de execução através da FADA-CHL.

Tabela 22: Execução da ferramenta FADA-CHL para o arquivo Lcet10.txt

N. Execução	HUFFMAN	LZW
1	21,536	1800,709
2	20,977	1875,722
3	20,992	1862,631
4	20,979	1805,765
5	21,009	1842,411
Média	**21,09**	**1837,44**

Fonte: do autor.

Para o arquivo Lcet10.txt obtemos como o tempo de execução médio com o algoritmo de Huffman o valor de 21,09 segundos e com o algoritmo de LZW o valor de 1837,44 segundos o algoritmo de HUFFMAN tem o tempo de execução melhor que o algoritmo de LZW.

6.2.8 Arquivo 8: Plrabn.txt

Escolhemos o arquivo Plrabn.txt que tem 481.861 bytes de informação, na tabela 23 são apresentados os resultados obtidos para o tempo de execução através da FADA-CHL.

Tabela 23: Execução da ferramenta FADA-CHL para o arquivo Plrabn.txt

N. Execução	HUFFMAN	LZW
1	19,086	1251,372
2	19,981	1298,529
3	19,057	1299,381
4	19,001	1301,98
5	19,4389	1288,357
Média	**19,31**	**1287,92**

Fonte: do autor.

Para o arquivo Plrabn.txt obtemos como o tempo de execução médio com o algoritmo de Huffman o valor de 19,31 segundos e com o algoritmo de LZW o valor

de 1287,92 segundos o algoritmo de HUFFMAN tem o tempo de execução melhor que o algoritmo de LZW.

6.2.9 Arquivo 9: Ptt5

Escolhemos o arquivo Ptt5 que tem 513.216 bytes de informação, na tabela 24 são apresentados os resultados obtidos para o tempo de execução através da FADA-CHL.

Tabela 24: Execução da ferramenta FADA-CHL para o arquivo Ptt5

N. Execução	HUFFMAN	LZW
1	0,295	374,748
2	0,234	386,422
3	0,279	379,353
4	0,280	373,22
5	0,307	399,157
Média	**0,27**	**382,58**

Fonte: do autor.

Para o arquivo Ptt5 obtemos como o tempo de execução médio com o algoritmo de Huffman o valor de 0,27 segundos e com o algoritmo de LZW o valor de 382,58 segundos o algoritmo de HUFFMAN tem o tempo de execução melhor que o algoritmo de LZW.

6.2.11 Arquivo 11: Xagrs.1

Escolhemos o arquivo Xagrs.1 que tem 4.227 bytes de informação, na tabela 25 são apresentados os resultados obtidos para o tempo de execução através da FADA-CHL.

Tabela 25: Execução da ferramenta FADA-CHL para o arquivo Xagrs.1

N. Execução	HUFFMAN	LZW
1	0,316	0,746
2	0,284	0,726
3	0,306	0,722
4	0,278	0,728
5	0,267	0,793
Média	**0,29**	**0,74**

Fonte: do autor.

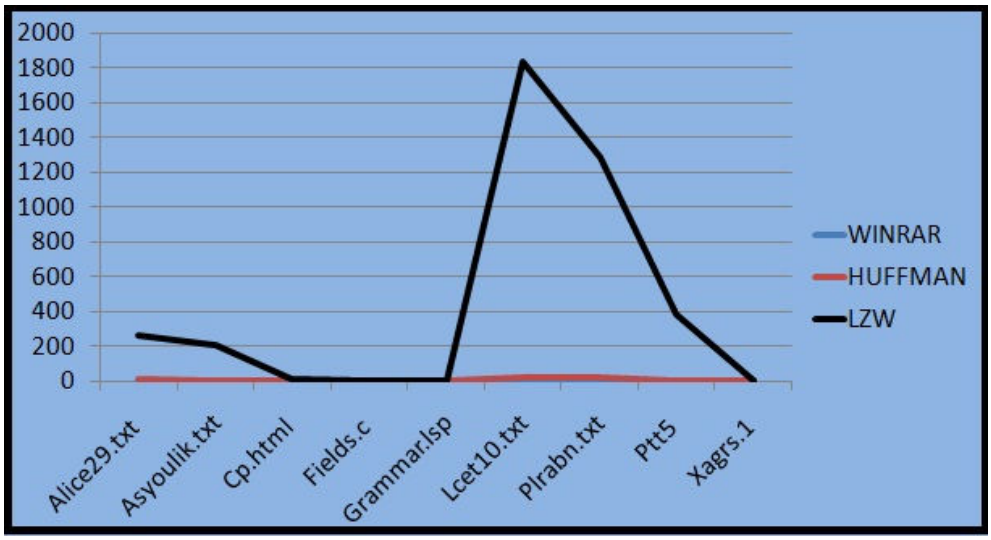
Para o arquivo Xagrs.1 obtemos como tempo de execução médio com o algoritmo de Huffman o valor de 0,29 segundos e com o algoritmo de LZW o valor de 0,74 segundos o algoritmo de HUFFMAN tem o tempo de execução melhor que o algoritmo de LZW.

Tabela 26: Tempos de compactação dos arquivos
do Canterbury Corpus.

Num. Arq	WINRAR	HUFFMAN	LZW
1	2,3	6,23	260,99
2	2,0	5,39	205,94
3	0,91	1,44	13,08
4	0,47	0,65	4,31
5	0,29	0,26	0,58
6	---------	----------	----------
7	2,04	21,09	1837,44
8	1,23	19,31	1287,92
9	0,98	0,27	382,58
10	---------	----------	----------
11	0,34	0,29	0,74
Total	**1,17**	**6,10**	**443,73**

Fonte: do autor.

Figura 23: Gráfico de tempos de compactação dos arquivos do Canterbury Corpus.



Fonte: do autor.

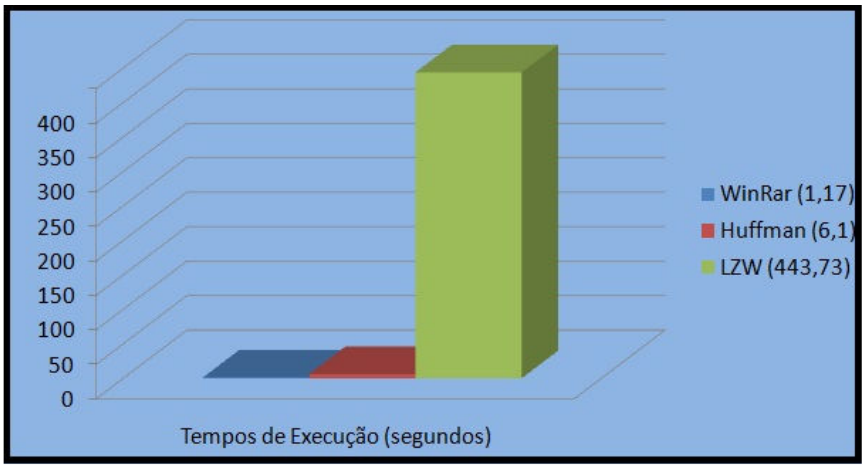
Na tabela 27 demonstraremos a velocidade média dos métodos, os tempos são relativos aos tempos de compactação do sistema WINRAR. Um fato interessante é que quanto menor a taxa de compactação de um algoritmo mais rápido pode ser sua execução, ou seja, seu tempo de execução é melhor, o que não significa que o algoritmo é o melhor.

Tabela 27. Velocidades médias de compactação dos métodos.

Método	**Compactação**
WINRAR	1,17
HUFFMAN	6,10
LZW	443,73

Fonte: do autor.

Figura 23. Gráfico de velocidades médias de compactação dos métodos.



Fonte: do autor.

6.3 MEMÓRIA

Como terceiro resultado nessa análise temos um fator muito importante na hora de escolher um método de compactação é a memória, ou seja, é a quantidade de memória utilizada para a execução dos algoritmos, exemplo desse recursos são as tabelas de freqüência utilizadas pelos métodos estatísticos, os elementos presentes nas arvores binárias de Huffman ou a tabela utilizada pelo algoritmo de LZW na montagem de seu dicionário de símbolos ou outros recursos necessários. Neste caso analisaremos nossos dois métodos separadamente.

6.3.1 Arquivo 1: Alice29.txt

Escolhemos o arquivo Alice29.txt que tem 152.089 bytes de informação, na tabela 29 são apresentados os resultados obtidos para o tempo de execução em segundos através da FADA-CHL.

Tabela 29: Execução da ferramenta FADA-CHL para o arquivo Alice29.txt

N. Execução	HUFFMAN	LZW
1	5,73	9
2	6,1	8,96
3	5,98	10,89
4	5,12	7
5	7,90	9,12
Média	**6,1**	**8,99**

Fonte: do autor.

Para o arquivo Alice29.txt obtemos como taxa média de uso de memória com o algoritmo de Huffman o valor de 6,1 megabytes e com o algoritmo de LZW o valor de 8,99 megabytes, o algoritmo de HUFFMAN tem taxa média de uso de memória melhor que o algoritmo de LZW.

6.3.2 Arquivo 2: Asyoulik.txt

Escolhemos o arquivo Asyoulik.txt que tem 125.179 bytes de informação, na tabela 30 são apresentados os resultados obtidos para o tempo de execução através da FADA-CHL.

Tabela 30: Execução da ferramenta FADA-CHL para o arquivo Asyoulik.txt

N. Execução	HUFFMAN	LZW
1	2,4	4,2
2	6,0	4,82
3	2,67	4,03
4	2,89	4,12
5	2,1	3,99
Média	**3,21**	**4,23**

Fonte: do autor.

Para o arquivo Asyoulik.txt obtemos como taxa média de uso de memória com o algoritmo de Huffman o valor de 3,21 megabytes e com o algoritmo de LZW o

valor de 4,23 megabytes, o algoritmo de HUFFMAN tem taxa média de uso de memória melhor que o algoritmo de LZW.

6.3.3 Arquivo 3: Cp.html

Escolhemos o arquivo Cp.html que tem 24.603 bytes de informação, na tabela 31 são apresentados os resultados obtidos para o tempo de execução através da FADA-CHL.

Tabela 31: Execução da ferramenta FADA-CHL para o arquivo Cp.html

N. Execução	HUFFMAN	LZW
1	4,04	6
2	3,99	5,98
3	3,67	5,45
4	4,09	6,12
5	4,01	6,34
Média	**3,96**	**5,97**

Fonte: do autor.

Para o arquivo Cp.html obtemos como taxa média de uso de memória com o algoritmo de Huffman o valor de 3,96 megabytes e com o algoritmo de LZW o valor de 5,97 megabytes, o algoritmo de HUFFMAN tem taxa média de uso de memória melhor que o algoritmo de LZW.

6.3.4 Arquivo 4: Fields.c

Escolhemos o arquivo Fields.c que tem 11.150 bytes de informação, na tabela 32 são apresentados os resultados obtidos para o tempo de execução através da FADA-CHL.

Tabela 32: Execução da ferramenta FADA-CHL para o arquivo Fields.c

N. Execução	HUFFMAN	LZW
1	6,8	4
2	6,91	4,12
3	6,98	3,99
4	6,01	4,14
5	5,99	4,72
Média	**6,35**	**4,19**

Fonte: do autor.

Para o arquivo Fields.c obtemos como taxa média de uso de memória com o algoritmo de Huffman o valor de 6,35 megabytes e com o algoritmo de LZW o valor de 4,19 megabytes, o algoritmo de LZW tem taxa média de uso de memória melhor que o algoritmo de HUFFMAN.

6.3.5 Arquivo 5: Grammar.lsp

Escolhemos o arquivo Grammar.lsp que tem 3.721 bytes de informação, na tabela 33 são apresentados os resultados obtidos para o tempo de execução através da FADA-CHL.

Tabela 33: Execução da ferramenta FADA-CHL para o arquivo Grammar.lsp

N. Execução	HUFFMAN	LZW
1	5,3	3
2	5,23	2,97
3	5,12	2,56
4	5,34	2,99
5	5,99	3,1
Média	**5,39**	**2,92**

Fonte: do autor.

Para o arquivo Grammar.lsp obtemos como taxa média de uso de memória com o algoritmo de Huffman o valor de 5,39 megabytes e com o algoritmo de LZW o valor de 2,92 megabytes, o algoritmo de LZW tem taxa média de uso de memória melhor que o algoritmo de HUFFMAN.

6.3.7 Arquivo 7: Lcet10.txt

Escolhemos o arquivo Lcet10.txt que tem 426.754 bytes de informação, na tabela 34 são apresentados os resultados obtidos para o tempo de execução através da FADA-CHL.

Tabela 34: Execução da ferramenta FADA-CHL para o arquivo Lcet10.txt

N. Execução	HUFFMAN	LZW
1	12	16,86
2	14,89	15,99
3	13,78	16,45
4	12,03	16,89
5	12,89	16,09
Média	**13,11**	**16,45**

Fonte: do autor.

Para o arquivo Lcet10.txt obtemos como taxa média de uso de memória com o algoritmo de Huffman o valor de 13,11 megabytes e com o algoritmo de LZW o valor de 16,45 megabytes, o algoritmo de HUFFMAN tem taxa média de uso de memória melhor que o algoritmo de LZW.

6.3.8 Arquivo 8: Plrabn.txt

Escolhemos o arquivo Plrabn.txt que tem 481.861 bytes de informação, na tabela 35 são apresentados os resultados obtidos para o tempo de execução através da FADA-CHL.

Tabela 35: Execução da ferramenta FADA-CHL para o arquivo Plrabn.txt

N. Execução	HUFFMAN	LZW
1	13,42	14,5
2	13,24	13,12
3	12,89	11,89
4	12,67	12,89
5	13,89	13,03
Média	**13,22**	**13,08**

Fonte: do autor.

Para o arquivo Plrabn.txt obtemos como o tempo de execução médio com o algoritmo de Huffman o valor de 19,31 segundos e com o algoritmo de LZW o valor de 1287,92 segundos o algoritmo de LZW tem o tempo de execução melhor que o algoritmo de HUFFMAN.

6.3.9 Arquivo 9: Ptt5

Escolhemos o arquivo Ptt5 que tem 513.216 bytes de informação, na tabela 36 são apresentados os resultados obtidos para o tempo de execução através da FADA-CHL.

Tabela 36: Execução da ferramenta FADA-CHL para o arquivo Ptt5

N. Execução	HUFFMAN	LZW
1	11,42	15,5
2	11,23	19,34
3	11,14	14,98
4	12,99	15,03
5	11,96	15,76
Média	**11,74**	**16,12**

Fonte: do autor.

Para o arquivo Lcet10.txt obtemos como taxa média de uso de memória com o algoritmo de Huffman o valor de 11,74 megabytes e com o algoritmo de LZW o valor de 16,12 megabytes, o algoritmo de HUFFMAN tem taxa média de uso de memória melhor que o algoritmo de LZW.

6.3.11 Arquivo 11: Xagrs.1

Escolhemos o arquivo Xagrs.1 que tem 4.227 bytes de informação, na tabela 37 são apresentados os resultados obtidos para o tempo de execução através da FADA-CHL.

Tabela 37: Execução da ferramenta FADA-CHL para o arquivo Xagrs.1

N. Execução	HUFFMAN	LZW
1	5,2	1
2	4,9	1,23
3	4,8	1,02
4	4,91	2,19
5	5,23	1,9
Média	**5,00**	**1,46**

Fonte: do autor.

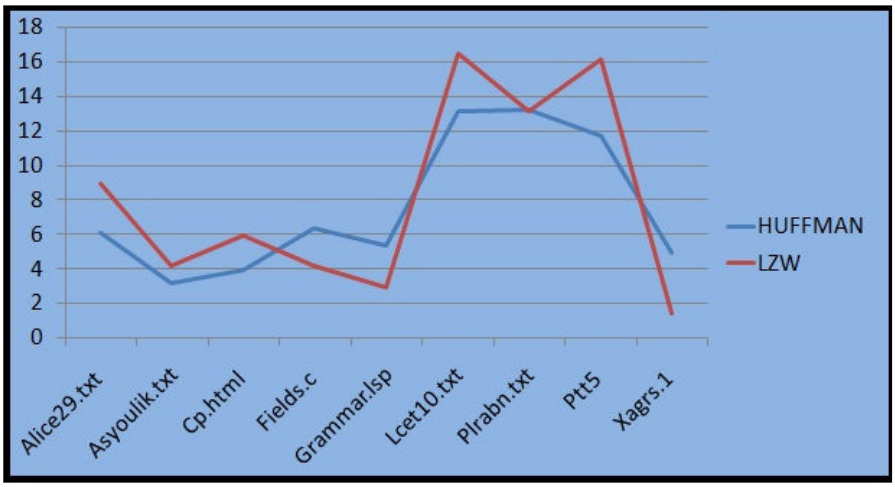
Para o arquivo Lcet10.txt obtemos como taxa média de uso de memória com o algoritmo de Huffman o valor de 5,00 megabytes e com o algoritmo de LZW o valor de 1,46 megabytes, o algoritmo de LZW tem taxa média de uso de memória melhor que o algoritmo de HUFFMAN.

Tabela 38: Taxas médias de uso de memória dos arquivos do Canterbury Corpus.

Num. Arq	HUFFMAN	LZW
1	6,1	8,99
2	3,21	4,23
3	3,96	5,97
4	6,35	4,19
5	5,39	2,92
6	----------	----------
7	13,11	16,45
8	13,22	13,08
9	11,74	16,12
10	----------	----------
11	5,00	1,46
Total	**7,56**	**8,15**

Fonte: do autor.

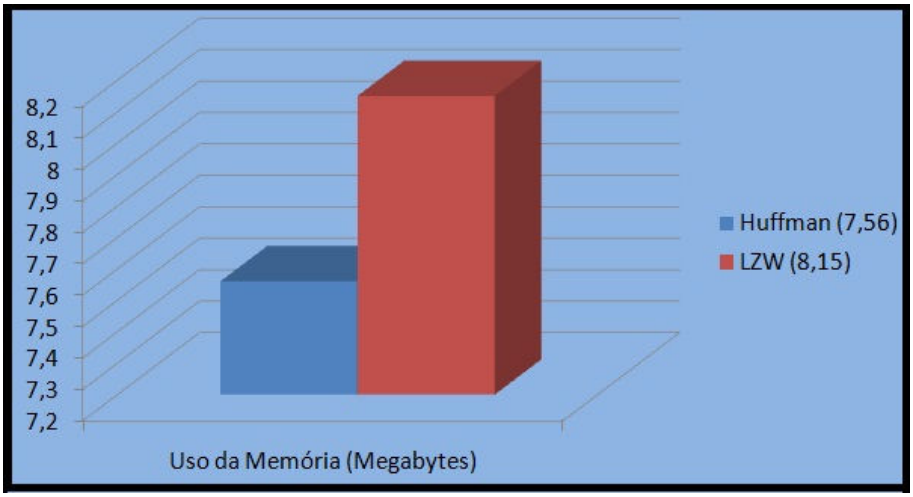
Figura 24: Gráfico de taxas médias de uso de memória dos arquivos do Canterbury Corpus.



Fonte: do autor.

Tabela 38: Taxas médias de uso de memória dos arquivos

do Canterbury Corpus.



Fonte: do autor.

Através do uso da ferramenta FADA-CHL obtemos as taxas médias de uso de memória, com isso podemos ver que em uma média geral os algoritmos aplicados a tabela Canterbury Corpus o algoritmo de HUFFMAN obteve uma melhor taxa de uso de memória.

7. TRABALHOS CORRELATOS

7.1 PROTÓTIPO DE SOFTWARE PARA OCULTAR TEXTOS COMPACTADOS EM ARQUIVOS DE ÁUDIO UTILIZANDO ESTEGANOGRAFIA

Trabalho desenvolvido por André Kobuszewski, mestrando em Ciência da Computação, pela Universidade Regional de Blumenau.

Este trabalho apresenta a especificação e implementação de um protótipo de software para incluir mensagens de texto compactadas em arquivos de áudio digitais, utilizando compactação de dados em conjunto com esteganografia. Utilizou-se do algoritmo de Huffman, técnica estatística de compactação de dados para a compactação de texos, em conjunto com o método de esteganografia *Last Significant Bit* (LSB), para ocultar mensagenscompactadas em arquivos de áudio no formato *Wave.*

7.2 IMPLEMENTAÇÃO DE UM ALGORITMO DE COMPACTAÇÃO EM HARDWARE

Trabalho desenvolvido por Tiago Lima de A. Fonseca, graduando em Engenharia da Computação, pela Universidade Federal do Pernambuco.

Neste trabalho apresentamos uma solução para compactação de arquivos, voltada para arquivos de imagem, usando *hardware* para realizar tal operação. Mostraremos um algoritmo implementado em VHDL para configuração de FPGAs. Como estudo de caso, aplicaremos o algoritmo implementado, em uma dada figura utilizando o conceito de compactação utilizando o algoritmo de Huffman.

7.3 DEVELOPMENT AND COMPARISON OF IMAGE ENCODERS BASED ON DIFFERENT COMPRESSION TECHNIQUES

Trabalho desenvolvido por Marc Rosanes Siscart, graduando em Engenharia da Computação, pela Universidade Politécnica da Catalunia.

Neste trabalho apresentamos uma solução para compactação de imagens sem perda de informação, demonstra o funcionamento de técnicas como JPG e o uso dos algoritmos de HUFFMAN, e a utilização desses algoritmos para redução de bits/bytes da informação.

8 CONCLUSÃO

Quando falamos em escolher um método ou outro método para compactação de dados concordamos que alguns parâmetros devem ser observados, a escolha em si não é muito simples, a escolha depende muito do tipo de arquivo que vamos compactar, ou o quando queremos reduzir desse arquivo durante a compactação, ou até mesmo quantos tempos têm para que essa operação seja executada. Os recursos disponíveis para essa operação quanto a espaço em disco, memória suficiente para executar a operação ou outros recursos computacionais. Percebemos através de testes que um determinado algoritmo pode ter uma alta taxa de compactação quando se trata de arquivos de texto, paginas de internet ou executáveis porem podem não ser tão bons com planilhas ou documentos do Microsoft Office.

Determinados métodos podem ser muito rápidos, mais quando se trata de taxa de compactação não alcançam valores expressivos, alguns algoritmos podem compactar arquivos rapidamente e com alta taxa de compactação mais utilizarem muitos recursos dos computadores. Como já foi citado no trabalho alguns algoritmos de compactação possuem métodos patenteados o que pode influenciar na escolha pois algoritmos da família LZW por exemplo patente registrada. Um dado importante, porém, é que, para aplicações estáticas, os códigos de HUFFMAN são melhores, ao passo que, para aplicações adaptativas e online, a LZW funciona melhor.

Referente ao dados coletados pela FADA-CHL em relação ao algoritmos HUFFMAN e LZW percebemos que o algoritmo de HUFFMAN foi superior a taxa de compactação aplicado a tabela CanterBury Corpus reduzindo cerca de 43,71% dos arquivos da tabela em média, contra uma redução média de 38,58% do algoritmo de LZW. Quando ao tempo de tempo de execução o algoritmo de HUFFMAN também se mostrou superior com um tempo médio de 6,10 segundos contra 443,73 do algoritmo de LZW. Podemos observar que quando tratamos de arquivos de texto os algoritmos LZW e HUFFMAN possuem uma taxa de compactação muito parecida, mais o tempo de execução do algoritmo de LZW é muito maior. Quanto a taxa de uso de memória através da ferramenta FADA-CHL podemos observar que o uso de memória dos dois algoritmos é parecido sendo que para alguns arquivos o Huffman usa mais memória, e para outros arquivos o LZW usa mais memória, essa elevações

na taxa de uso estão relacionadas a construção dos algoritmos em relação ao uso de dicionário de dados ou arvores binárias.

Ao cruzarmos os dados com os obtidos através do software WINRAR percebemos que ele busca obter menores tempos de execução possuindo uma taxa de compactação maior, ele consegue isso pois possui na sua estrutura mais de um algoritmo de compactação, possuindo um método de escolha do melhor em relação ao tipo de entrada selecionado. Os arquivos 6 e 10 da tabela CanterBury Corpus foram removidos do teste pois possuímos dificuldades na implementação do software de comparação para as extensões xls e sum.

Como uma conclusão podemos dizer que quando aplicados a tabela CanterBury o algoritmo de HUFFMAN é mais eficiente em todos os parâmetros abordados e a ferramenta desenvolvida atende os requisitos e funcionalidades para a qual foi desenvolvida. Como trabalhos futuros sugerimos a implementação de análise estatística sobre os resultados obtidos e fazer com que a própria ferramenta possa gerar esses resultados, desenvolver a parte de descompactação e gerar uma comparação entre os resultados obtidos na compactação e descompactação dos arquivos, cruzando as informações e obtendo os resultados de cada um, desta forma obtendo o melhor algoritmo dentro das duas finalidades.

REFERÊNCIAS

BUSCHART, Cameron. ***File Compression.*** Disponível em: <http://chesworth.com/pv/technical/file_compression.htm#duh >. Acesso em: 23 mar. 2012.

FRANÇA NETO, Leopoldo Rodrigues. ***Um Ambiente para Processamento de Grandes Acervos de Imagens.*** Departamento de Computação, Universidade Federal de Pernambuco. Disponível em: < http://www.netpe.com.br/~leopoldo/tese/ >. Acesso em: 23 mar. 2012.

GANE, Chris; SARSON, Trish. ***Análise estruturada de sistemas,*** **tradução Gerry Edward Tompkins.** Rio de Janeiro – RJ, Editora S.A. 1983. 257p.

HELD, Gilbert; MARSHALL, Thomas R. ***Compactação de dados: Técnicas e aplicações, considerações de Hardware e Software.*** Trad. Andreia Dell'more Santos. Edição 10 Tatuapé-SP. Livros Érica Editora Ltda. 1998. 390p.

KRISTULA, ***The Compression Site****.* Disponível na Internet: < http://www.davesite.com/computers/compress.shtml>. Acesso em: 23 mar. 2012.

KUMPF, Kai. ***Lempel-Ziv Compression and Complexity.*** Disponível na Internet: <http://genome.imb-jena.de/~kku/lztop.html >. Acesso em: 23 mar. 2012.

LAKATOS, Eva Maria; MARCONI, Marina de Andrade. ***Metodologia do Trabalho Cientifico.*** Edição 4. São Paulo - SP. Editora Atlas S.A., 1994. 214p.

MULLER, Daniel Nehme. ***Compactação de dados.*** Disponível na Internet: < http://www.ulbra.tche.br/~danielnm/ed/E/polE.html>. Acesso em: 23 mar. 2012.

NELSON, Mark. ***LZW Data Compression.*** Disponível na Internet : <http://www.dogma.net/markn/articles/lzw/lzw.htm.>. Acesso em: 23 mar. 2012.

PINTO, Wilson Silva. ***Introdução ao desenvolvimento de algoritmos e estrutura de dados.*** São Paulo - SP. Edição 6a. Editora Érica Ltda. 1990, 152p.

PROSISE, Jeff. ***Como é feita a compactação de dados.*** Disponível na Internet: <http://www.portal.inf.br/materias.htm>. Acesso em: 23 mar. 2012.

RUIZ, Vicente Gonzáles. ***Una Implementación del Algoritmo de Compresión Lempel-Ziv Welch.*** Disponível na Internet: < http://www.ace.ual.es/~vruiz/investigacion/Tenerife-95/html/node1.html>. Acesso em: 23 mar. 2012.

SCHEIDEMANTEL, Rogério. ***Ferramentas para o Estudo da Aplicação de Compactação de Imagens em Processos Criptográficos.*** Dissertação de Mestrado. Departamento de Engenharia Elétrica, Universidade de Brasília - DF, 1995 117p.

SZWARCFITER, Jayme Luiz; MARKENZON, Lilian. ***Estruturas de dados e seus Algoritmos****.* Editora LTC-Livros Técnicos e Científicos. Rio de Janeiro-RJ. 1994. 324p

VILLAS, Marcos Vianna [et ai]. ***Estrutura de dados: conceitos e técnicas de implementação****. 2* Edição. Rio de Janeiro - RJ. Editora Campus. 1993, 296p

APÊNDICE(S)

APÊNDICE A – Artigo FADA-CHL: ferramenta para análise de desempenho e comparação dos algoritmos de compactação sem perda HUFFMAN e LZW.

FADA-CHL: FERRAMENTA PARA ANÁLISE DE DESEMPENHO E COMPARAÇÃO DOS ALGORITMOS DE COMPACTAÇÃO SEM PERDA HUFFMAN E LZW

Bruno Casemiro Kurzawe[1], Christine Vieira Scarpato[2]

[1]Curso de Ciência da Computação - Universidade do Extremo sul Catarinense (UNESC) - Criciúma - SC - Brasil

[2]Departamento de Ciência da Computação - Universidade do Extremo sul Catarinense (UNESC) - Criciúma - SC - Brasil

bruno.kurzawe@hotmail.com,cvs@unesc.net

Resumo. *A área de compactação de dados é uma das áreas mais antigas da computação, em 1950 já se estudavam métodos para isso. Nas décadas de 80 e 90 essa área teve um grande crescimento com o desenvolvimento de algoritmos que utilizamos até hoje como Huffman, LEMPEL-ZIV & WELCH LZW, Burrows-Wheeler Transform BWT e Prediction by Partial Matching PPM. Após ler sobre o assunto, e descobrir que compressores atuais como WinZip e WinRar utilizam esses algoritmos nos interessamos a conhecer melhor seu funcionamento, e através de técnicas de análise de desempenho, como a taxa de compactação, velocidade de compactação e uso da memória, descobrir o melhor algoritmo para cada tipo de arquivo. Neste trabalho implementamos uma ferramenta chamada FADA-CHL que implementa o algoritmo de HUFFMAN e o algoritmo de LZW, essa ferramenta analisa os resultados para os arquivos do benchmark da Canterbury Corpus desenvolvido pela universidade de CanterBury da Nova Zelandia. Durante o desenvolvimento da ferramenta obtivemos problemas com dois arquivos para analise por causa do tamanho desses arquivos.Através da ferramenta podemos concluir que quando aplicado a tabela CanterBury Corpus o algoritmo de HUFFMAN é mais eficiente devido a constituição dos arquivos.*

Palavras-chave*: Compactação de dados, algoritmos de compactação, análise de desempenho, taxa de compactação e tempo de execução.*

INTRODUÇÃO

Com a expansão da transmissão de dados em todo o mundo, o crescimento das redes de computadores e a expansão da internet, todos os dias surgem novas técnicas de processamento de dados a fim de encontrar melhores maneiras de transmiti-los ou armazená-los. HELD (1992) e FRANÇA NETO (1998) afirmam em suas obras que ao mesmo tempo em que a capacidade de transmissão e armazenamento cresce, as informações a serem transmitidas e armazenadas

crescem mais rápido ainda, desta forma a compactação de dados se faz útil e muito necessária.

Ao citarmos compactação e transmissão de dados logo consideramos que sua aplicação é muito importante em arquivos de imagem e vídeo, pois nesse tipo de arquivo encontramos uma quantidade grande de informação, tornando-os arquivos de tamanho elevado e de alto custo para sua transmissão e armazenamento. Para isso buscamos na compactação de dados uma maneira de solucionar esse problema. Dentro da compactação de dados abordamos dois tipos, compactação com perda e compactação sem perda de informações. Neste projeto o foco será a aplicação dos algoritmos de compactação sem perda, será feita uma análise comparativa entre os dois principais algoritmos utilizados nessa área, que são o algoritmo de HUFFMAN e o algoritmo LEMPEL-ZIV & WELCH LZW que é chamado de LZW.

COMPACTAÇÃO DE DADOS

De acordo com FRANÇA NETO (1998) a compactação de dados tem como intuito tentar reduzir ao máximo o espaço ocupado em um dispositivo de armazenamento de dados, por exemplo, discos rígidos ou pendrives. Esse tipo de operação é feita através de algoritmos de compactação de dados no qual o objetivo é reduzir a quantidade de bytes utilizados para representar determinados tipos de arquivos. Essa compactação de dados tem como objetivo reduzir a redundância de informação utilizada para representar um arquivo. Na figura 1 temos um esquema que FRANÇA NETO (1998) utilizou para representar a compactação e descompactação de determinado arquivo “D” onde este arquivo é comprimido se tornando o arquivo “C” e depois descomprimido voltando a ser o arquivo “D”.

A partir das definições HELD (1992) podemos dizer que atualmente a compactação de dados está presente nas aplicações de comunicação, como redes de computadores, onde ela é utilizada em protocolos de comunicação para reduzir as informações na hora de enviá-las. Vários padrões de compactação estão presentes no nosso dia-dia, tais como o ZIP, RAR entre outros. A compactação de dados possui inúmeras aplicações podendo ser utilizada tanto no nível de hardware quanto no nível de software.

Dentre as técnicas de compactação de dados podemos classificá-las em compactação com perda e compactação sem perda.

HELD (1992) afirma que algoritmos de compactação com perda são conhecidos como quantizadores, neste tipo de compactação o dado original é processado através de um método de quantização, permitindo altas taxas de redução do tamanho original. Porém alta taxa de redução de tamanho tem um custo, que seria a perda de informação em relação ao dado original, o arquivo perde a fidelidade em relação ao arquivo original.

Seguindo as afirmações de HELD (1992) podemos dizer que algoritmos de compactação sem perda são conhecidos como codificadores universais, esse tipo de codificador mantém a fidelidade do arquivo em relação ao arquivo de origem. Porém essa fidelidade tem um custo, em que as taxas de redução são diminuídas, ou seja, quando utilizamos compressores que utilizam métodos sem perda obtemos taxas de compactação menores, os arquivos compactados possuem tamanho modestamente menor. Isso se dá devido ao limite entrópico, o limite entrópico é a quantidade de caracteres diferentes presentes dentro da cadeia de símbolos, a entropia pode ser caracterizada como a quantidade real de dados de uma informação existente.

ALGORITMO DE HUFFMAN

O algoritmo de Huffman é segundo FRANÇA NETO (1998) um dos algoritmos de compactação sem perda mais conhecido, inicialmente este método foi aplicado à compactação de arquivos de texto onde buscava obter uma taxa de compactação ótima. David Huffman desenvolveu esta técnica que visava utilizar árvores binárias a fim de gerarem códigos binários. Dentro desta técnica se suponha que o arquivo de texto a ser compactado fosse composto por uma seqüência de caracteres ou símbolos, conhecendo a freqüência com que cada símbolo aparece seria possível atribuir um valor para cada símbolo através do número de ocorrências, desta forma compactando o arquivo inteiro. FRANÇA NETO (1998, 67p) afirma em sua obra que, "O algoritmo de Huffman é um aperfeiçoamento do algoritmo de Shannon-Fano, desenvolvido em 1952 por David Huffman. Desde então, surgiram diversas variações do algoritmo", apenas o algoritmo de Huffman foi desenvolvido

por David Huffman, o Shannon-Fano foi desenvolvido por por Claude Shannon e Robert Fano.

Seguindo as afirmações do autor o algoritmo de Huffman revolucionou a compactação quando foi criado, se tornou um dos algoritmos mais conhecidos sendo implementado por universidades pelo mundo todo. Seguindo a mesma lógica do algoritmo de compactação de Shannon-Fano o algoritmo de Huffman tem como objetivo principal a codificação dos símbolos que aparecem com uma freqüência maior com um número menor de bits e aqueles que aparecem com menos freqüência com um número maior de bits. Com isso podemos afirmar que as taxas de compactação através desse algoritmo podem ser maiores ou menores dependendo da probabilidade da distribuição dos símbolos dentro de um arquivo. A compactação através deste algoritmo é dada através de dois passos, no primeiro determinamos a forma que a probabilidade é distribuída entre os símbolos da fonte, utilizamos essa distribuição para a geração da tabela de códigos. No segundo passo é feita uma varredura na tabela gerada anteriormente, desta forma determinamos a codificação e as fontes reduzidas, apões determinarmos as codificações a serem utilizadas o arquivo por fim é compactado.

Segundo FRANÇA NETO (1998) a diferença entre o algoritmo de Shannon-Fano e o de Huffman está basicamente na maneira em que a árvore binária é construída, a árvore de decisão de Huffman não é gerada baseando-se em divisões de dois grupos de símbolos que é utilizada pelo algoritmo de Shannon-Fano, no qual a soma das probabilidades tem que ser igual ou semelhante, mais sim na soma das menores freqüências de símbolos encontradas nos arquivos de entrada.

ALGORITMO DE LZW

De acordo com FRANÇA NETO (1998) com o surgimento da compactação baseada em dicionário de dados, também conhecida como cadeia de compactação, a compactação de dados teve uma revolução por todo o mundo, a maioria dos aplicativos de compactação utilizados hoje utilizam algum algoritmo da família LZW criados por Lempel-Ziv & Tery Welch.

Segundo o autor o algoritmo de Lempel-Ziv & Welch é uma variação do algoritmo Lempel-ziv32, este algoritmo foi desenvolvido pelo inglês Tery Welch em

1984, a contribuição de Tery Welch no desenvolvimento do aprimoramento do algoritmo de Lempel-Ziv32 foi tão significativa e importante que existe uma tendência de se desconsiderar o algoritmo original desenvolvido por Tery Welch, por causa da melhora significativa que ele gerou ao algoritmo.

De acordo com WELCH (1984) o aprimoramento do método trabalha da seguinte forma: O algoritmo de Lempel-Ziv32 conhecido com LZ77 utilizava uma janela deslizante para codificar os blocos de caracteres iguais a fim de gerar um código de saída e esse código é o próprio dicionário de strings. O algoritmo trabalha direcionado aos blocos de caracteres mais recentes localizados dentro da janela. Por esse problema Tery Welch no ano seguinte abandonaram o conceito de janela deslizante e criaram o LZ7833 e começaram a utilizar o dicionário como uma lista ilimitada de blocos, onde os novos blocos, ou novas frases são inseridas em um dicionário e nas próximas vezes que esse bloco aparecer na fonte é gerado um código que as representa no dicionário.

Mesmo Terry Welch sendo o criador do método de compactação LZW ele não é o proprietário da patente, de acordo com a internet espanhola, na época que ele desenvolveu o algoritmo ele trabalhava como pesquisador na Sperry, uma empresa americana que adquiriu os direitos do algoritmo de LZW em 1981, a empresa patenteou o algoritmo, a empresa se fundiu com a Burroughs posteriormente. Atualmente a dona da patente é a UNISYS e ela delimita que todos precisam comprar uma licença para utilizar esse método em aplicações comerciais.

CONCLUSÕES

Referente ao dados coletados pela FADA-CHL em relação ao algoritmos HUFFMAN e LZW percebemos que o algoritmo de HUFFMAN foi superior a taxa de compactação aplicado a tabela CanterBury Corpus reduzindo cerca de 43,71% dos arquivos da tabela em média, contra uma redução média de 38,58% do algoritmo de LZW. Quando ao tempo de tempo de execução o algoritmo de HUFFMAN também se mostrou superior com um tempo médio de 6,10 segundos contra 443,73 do algoritmo de LZW. Podemos observar que quando tratamos de arquivos de texto os algoritmos LZW e HUFFMAN possuem uma taxa de compactação muito parecida, mais o tempo de execução do algoritmo de LZW é muito maior. Quanto a taxa de uso de memória através da ferramenta FADA-CHL podemos observar que o uso de

memória dos dois algoritmos é parecido sendo que para alguns arquivos o Huffman usa mais memória, e para outros arquivos o LZW usa mais memória, essa elevações na taxa de uso estão relacionadas a construção dos algoritmos em relação ao uso de dicionário de dados ou arvores binárias.

Ao cruzarmos os dados com os obtidos através do software WINRAR percebemos que ele busca obter menores tempos de execução possuindo uma taxa de compactação maior, ele consegue isso pois possui na sua estrutura mais de um algoritmo de compactação, possuindo um método de escolha do melhor em relação ao tipo de entrada selecionado. Os arquivos 6 e 10 da tabela CanterBury Corpus foram removidos do teste pois possuímos dificuldades na implementação do software de comparação para as extensões xls e sum.

Como uma conclusão podemos dizer que quando aplicados a tabela CanterBury o algoritmo de HUFFMAN é mais eficiente em todos os parâmetros abordados e a ferramenta desenvolvida atende os requisitos e funcionalidades para a qual foi desenvolvida. Como trabalhos futuros sugerimos a implementação de análise estatística sobre os resultados obtidos e fazer com que a própria ferramenta possa gerar esses resultados, desenvolver a parte de descompactação e gerar uma comparação entre os resultados obtidos na compactação e descompactação dos arquivos, cruzando as informações e obtendo os resultados de cada um, desta forma obtendo o melhor algoritmo dentro das duas finalidades.

REFERÊNCIAS

BUSCHART, Cameron. **File Compression.[on line] 1997**. Disponível em: <http://chesworth.com/pv/technical/file_compression.htm#duh >. Acesso em: 23 mar. 2012.

FRANÇA NETO, Leopoldo Rodrigues. **Um Ambiente para Processamento de Grandes Acervos de Imagens.** Departamento de Informática, Universidade Federal de Pernambuco. Disponível em: < http://www.netpe.com.br/~leopoldo/tese/ >. Acesso em: 23 mar. 2012.

HELD, Gilbert; MARSHALL, Thomas R. **Compactação de dados: Técnicas e aplicações, considerações de Hardware e Software.** Trad. Andreia Dell'more Santos. Edição 10 Tatuapé - SP. Livros Érica Editora Ltda. 1998. 390p.

MULLER, Daniel Nehme. **Compactação de dados.** Disponível na Internet: < http://www.ulbra.tche.br/~danielnm/ed/E/polE.html>. Acesso em: 23 mar. 2012.

NELSON, Mark. **LZW Data Compression.** Disponível na Internet : <http://www.dogma.net/markn/articles/lzw/lzw.htm.>. Acesso em: 23 mar. 2012.