UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

COMPRESSÃO DE ÁUDIO

ENDRIZZO GALILEI CARMO BRAGA

BRASÍLIA, 17 DE FEVEREIRO DE 2003

UNIVERSIDADE DE BRASÍLIA

FACULDADE DE TECNOLOGIA

DEPARTAMENTO DE ENGENHARIA ELÉTRICA

COMPRESSÃO DE ÁUDIO

POR

ENDRIZZO GALILEI CARMO BRAGA

PROJETO FINAL DE GRADUAÇÃO SUBMETIDA AO DEPARTAMENTO DE ENGENHARIA ELÉTRICA DA FACULDADE DE ENGENHARIA ELÉTRICA DA UNIVERSIDADE DE BRASÍLIA COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE ENGENHEIRO

APROVADA POR:

_____

JULIANA FERNANDES CAMAPUM WANDERLEY, PhD, UnB, (ORIENTADORA)

_____

LEONARDO RODRIGUES ARAUJO XAVIER DE MENEZES; PhD, UnB, (EXAMINADOR)

_____

LÚCIO MARTINS DA SILVA; DOUTOR;UnB (EXAMINADOR)

BRASÍLIA, 17 DE FEVEREIRO DE 2003.

FICHA CATALOGRÁFICA

BRAGA, ENDRIZZO GALILEI CARMO
Compressão de Áudio [Distrito Federal] 2003
(xv), (76)p., 297 mm (ENE/FT/UnB, Engenheiro, Engenharia Elétrica, 2003).

Projeto Final de Graduação – Universidade de Brasília, Faculdade de Tecnologia.
Departamento de Engenharia Elétrica.

1. Telecomunicações 2. Processamento de Sinais
3. Compressão de Áudio

I. ENE/FT/UnB. II. Título (Série)

REFERÊNCIA BIBLIOGRÁFICA

BRAGA, ENDRIZZO GALILEI CARMO (2003). Compressão de Áudio. (Projeto Final de Graduação), Publicação Fevereiro/2003, Departamento de Engenharia Elétrica, Universidade de Brasília, Brasília, DF, (76)p.

CESSÃO DE DIREITOS

NOME DO AUTOR: Endrizzo Galilei Carmo Braga
TÍTULO DA DISSERTAÇÃO: COMPRESSÃO DE ÁUDIO
GRAU/ANO: Engenheiro/2002.

_____

ENDRIZZO GALILEI CARMO BRAGA

AOS 8 BLOCO E APTº 401

CEP 70.660-085 – CRUZEIRO – DF – BRASIL

“Sonho que se sonha só, é só um sonho só”.

Sonho que se sonha junto é realidade “

Raul Seixas

AGRADECIMENTOS

Gostaria de agradecer primeiramente a Deus que nunca me deixou desamparado e sempre me mostrou o caminho certo a seguir.

A meus pais, Maria e Braga; a meus irmãos, Régia e Ata, e a minha namorada Franciele que sempre me apoiaram e são os grandes responsáveis por minhas vitórias. Serei eternamente grato.

A meus Tios, Marilda, Wellington, Braguinha, Guiomar, Helena e Biju pelos quais tenho grande estima e admiração.

Aos meus primos, Beto, Laura, Rafael, Mayara, Paulinho, Dú Carmo, Hévila e Thiago pela ajuda e pelos momentos de alegria que me proporcionaram.

Aos meus cunhados e amigos, Júnior, Andréia, Eduardo, Fabrício, Diórgenes, Daniel Araújo, Daniel Andrade, Gustavo, Isabel, Aritan, Plínio, Artur, Michelle, Domingos e a todos que me ajudaram nessa caminhada.

Também gostaria de agradecer ao Jefferson e Cláudio pela paciência e a todos que trabalham comigo no Departamento de Engenharia da ITSA.

Um agradecimento especial a minha orientadora Juliana, a qual tenha muita consideração, pela ajuda sem a qual não teria conseguido completar esse trabalho.

RESUMO

Atualmente devido aos avanços tecnológicos e o aumento do tráfego nas redes de comunicações é quase impossível imaginar o cotidiano sem a compressão de dados. Isso se deve principalmente a necessidade crescente de transmitir ou armazenar grande quantidade de informação.

Por isso, com o passar dos anos, surgiu a necessidade de se utilizar menos dados para representar uma certa informação contudo, não se deve suprimir informação relevante. Foi nesse contexto que nasceu a compressão de dados, que se pode definir como a ciência de representar uma mesma informação utilizando uma menor taxa de bits.

Esse trabalho se propõe a demonstrar a importância da compressão de áudio fornecendo soluções para a melhoria do fluxo de dados, como é o caso das técnicas de compressão de áudio Codificação de Huffman, DPCM e LPC, que serão abordadas nesse trabalho.

Com a utilização dessa metodologia pretende-se minimizar os esforços computacionais nas tarefas de armazenamento e transmissão de dados através da compressão da informação, aumentando o desempenho e facilitando a execução de serviços de comunicação em tempo real.

ABSTRACT

Nowadays due to the technology advances and the increase in network traffic, it is almost impossible to imagine the daily life without data compression. The main reason is the increasing need to transmit and store huge amount of information

Consequently, as the years go by, the need to use less data to represent a given information has arisen. However, one should not suppress relevant information. In this context, the idea of data compression was born, which can be defined as the science that represent the same information with a smaller bit rate.

This work proposes to demonstrate the importance of audio compression providing solutions to improve data flow. The data compression techniques called Huffman Coding, DPCM and LPC, will be studied and tested.

By using this methodology we intend to minimize the burdensome computational requirements for storage and data transmission through information compression, thus increasing the performance and making easier the implementation of communication services in real time.

SUMÁRIO

ÍNDICE DE FIGURAS

ÍNDICE DE TABELAS

1. Introdução

Nas últimas décadas nós estamos passando por uma verdadeira revolução nas telecomunicações, que engloba desde o crescimento da Internet, com o advento do ADSL e cable modems, ao surgimento da comunicação móvel. Estas novas tecnologias aumentaram a quantidade de informação transformando a sociedade da era industrial para a era da informação. O aumento na quantidade de informação aumentou a importância dos algoritmos de compressão de dados como imagens, áudio e vídeo.

Atualmente, esses algoritmos estão em quase todas atividades do cotidiano humano. Por exemplo, quando você faz uma transmissão de dados de longa distância, ou se utiliza rede de telefonia pública através do modem ou conexões via satélite. Em ambos os casos, a compressão está presente. E é essa disseminação de novas tecnologias, das quais a compressão de dados faz parte, que denominamos de revolução.

A compressão pode ser definida a partir desse trecho escrito por Sayood [1]: “Compressão, redução da taxa de bits e redução de dados têm basicamente o mesmo significado, ou seja, quer se obter a mesma informação porém utilizando menor quantidade de dados. Há vários motivos para que a compressão se torne popular. Podemos citar a diminuição da quantidade de memória necessária para o armazenamento (hardware menor), o menor custo devido a uma menor largura de banda requerida para a transmissão da mesma quantidade de dados e a facilidade nas transmissões em tempo real.”

Os avanços nas técnicas de compressão são decorrentes dos avanços na teoria de processamento de sinais. Estas técnicas exigem grande esforço computacional o que está sendo solucionado com o avanço da microeletrônica através dos circuitos integrados para aplicações específicas (ASICs), chips de processamento de sinais (DSPs) e microprocessadores de grande capacidade.

Outro fator que está facilitando a incorporação da tecnologia de compressão de dados é a incrível corrida em direção a padronização. A padronização é importante para que o destinatário possa compreender como decodificar o conteúdo que está recebendo. Assim, para que a comunicação se estabeleça é necessário adotar o mesmo padrão na origem e no destino. Existem diversos padrões de codificação de áudio sendo que a preponderância de um em relação a outros pode ser por questões técnicas ou mercadológicas. O padrão PCM para digitalização de voz será apresentado no capítulo 6. Este padrão é utilizado por empresas prestadoras de serviços de telecomunicações.

Entretanto, apesar dos avanços tecnológicos, a quantidade de informação a ser transmitida aumentou mais do que a capacidade de compressão gerando problemas de tráfego nas redes de comunicação pois os meios de transmissão não estão preparados para suportar tamanho fluxo de informação.

Portanto, como se explica o fato do foco das pesquisas não estar concentrado no desenvolvimento de melhores tecnologias de transmissão e armazenamento? Isso até que acontece, mas não na rapidez necessária para acompanhar as inovações. Por exemplo, as aplicações que transmitem grandes volumes de informação e dispensam o uso da compressão, como é o caso do CD-ROM, do ADSL (Asymmetric Digital Subscriber Lines) e do Cable Modem. Existem situações em que a capacidade de transmissão não cresce significativamente por causas de agentes externos ao processo que limita o seu avanço, como é o caso da atmosfera.

É nesse cenário que se insere a compressão de dados, que pode ser denominada como a ciência da representação da informação em formas compactas. Essas representações compactas são frutos de análises na estrutura dos dados.

Um dos mais antigos exemplos de compressão de dados é o Código Morse desenvolvida por Samuel Morse no meio do século XIX. Observando o envio de telegramas, Morse constatou que todas letras possuíam um código, mas que havia letras que ocorriam mais freqüentemente do que outras. No intuito de diminuir o tempo médio de envio de uma mensagem, ele determinou que letras que ocorressem mais freqüentemente fossem definidas por menores seqüências do que letras que não aparecem freqüentemente numa mensagem. Essa mesma idéia é utilizada na codificação de Huffman estudada no capítulo 5. Portanto, há muitas estruturas presentes numa certa informação que podem ser utilizadas na compressão.

Para se ter uma idéia da importância da compressão de áudio preste atenção nesses cálculos. O áudio codificado a uma freqüência de amostragem de 44.1 kHz, 16 bits por amostra, estéreo (qualidade de CD) usa 44100 x 16 x 2 = 1.411.200 bits por segundo. Isto significa que para transmitir em tempo real tal arquivo por uma rede, é necessária uma largura de banda de 1,41 Mbits/s [2]. Uma música de três minutos (180 segundos) de áudio nestas condições usa 180x44100x16x2 = 254.016.000 bits, ou mais de 30 MB de armazenamento no computador. Através destes exemplos podemos verificar a importância da compressão do áudio.

É preciso satisfazer certas condições para se proceder à compressão, entre as quais a obrigatoriedade do sinal original estar representado em sua forma digital, que além de tornar possível a compressão de áudio, acarreta outras vantagens como:

- Segurança dos dados, pois facilita a criptografia;
- Menor sensibilidade a ruído e maior facilidade na regeneração do sinal;
- Maior propensão à detecção de erros;
- Universalização da representação.

Este trabalho tem como intuito apresentar um estudo sobre compressão de dados, abordando dois métodos existentes: compressão com perdas e sem perdas. Para melhor entendimento, o conteúdo foi dividido em oito capítulos.

No Capítulo 2 são abordadas a perceptividade auditiva e a estrutura do ouvido humano com o intuito de demonstrar a sensibilidade e o funcionamento desse órgão humano.

Em seguida, no Capítulo 3, é apresentado o fundamento da digitalização de um sinal, ou seja, são estudados os processos de amostragem, de quantização uniforme e não-uniforme e de codificação.

A abordagem do Capítulo 4 é sobre os métodos de compressão existentes e maneiras de quantificar a desempenho tanto quantitativamente como qualitativamente. Além disso deu-se uma breve argumentação sobre a teoria da informação.

A compressão sem perdas, seus modelos e o exemplo do método de codificação de Huffman está no quinto capítulo, onde é apresentado todo o procedimento intrínseco ao método, tanto na sua etapa de codificação quanto na de decodificação do sinal.

No sexto capítulo, a compressão com perdas é introduzida. Da mesma forma como na ocasião da compressão sem perdas, também é apresentado um exemplo de compressão, nesse caso a codificação DPCM e LPC.

Passando-se para uma abordagem mais empírica alguns resultados foram demonstrados no sétimo capítulo. Estes resultados são provenientes de testes com algoritmos que utilizam os métodos da codificação Huffman e da codificação DPCM.

Finalmente, no oitavo capítulo apresentamos as conclusões sobre a tecnologia de compressão de dados e sobre a performance dos algoritmos estudados.

2. Percepção Auditiva

Os ouvidos respondem a milhares de vibrações diferentes e, por meio do nervo auditivo, mandam sinais nervosos até o cérebro, que reconhece os sons. Como o ser humano possui dois ouvidos existe, além do reconhecimento dos sons, a identificação espacial da origem. Isso ocorre devido a diferença de tempo entre a chegada das vibrações sonoras em cada um dos ouvidos [3].

As orelhas, localizadas na parte externa, agem como um funil, captando o som e enviando-o por um canal até o tímpano, uma membrana que recobre o canal e que por ser muito fino vibra até por causa de ondas sonoras de pequenas amplitudes.

Essas vibrações fazem com que um pequeno osso, chamado martelo, pressione outro osso, o estribo. Dessa forma, as vibrações são ampliadas e transmitidas ao ouvido interno, ou cóclea, que possui um líquido que facilita a transmissão do som. Quando as ondas sonoras passam pela cóclea, elas agitam os pêlos, e essa agitação estimula minúsculas células nervosas que mandam sinais ao cérebro pelo nervo auditivo.

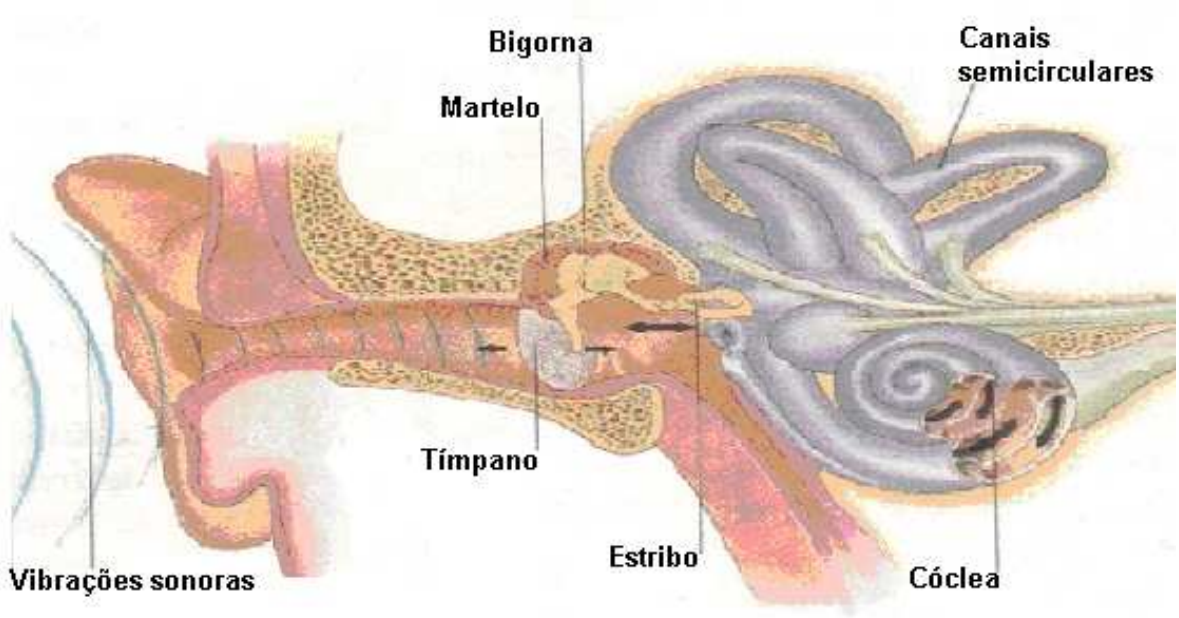

Figura 2.1 – Ouvido Humano

O ouvido humano pode distinguir sons na faixa de freqüência entre 20 Hz a 20 kHz. A extensão dessa faixa de freqüência decresce com a idade, ou seja, pessoas idosas são incapazes de captar as altas freqüências.

A audição humana não é perfeita. Além das limitações físicas do ouvido, depois o som tem que viajar através dos nervos até o córtex auditivo do cérebro onde ele é transformado em diferentes percepções das quais tomamos consciência.

O fato do ouvido humano não poder detectar a direção das baixas freqüências é chamado Redundância Estereofônica. Todas estas imperfeições e limitações na audição tornam possível a redução na quantidade de informação de áudio, sem afetar o que ouvimos. Em seguida, será discutida a sonoridade dos sons.

2.1 Sonoridade (Loudness)

Dois sons com a mesma amplitude podem ser percebidos com intensidade distinta dependendo das suas freqüências. A percepção da intensidade de um som não é constante com a freqüência. Observando a curva da Figura 2.2, vê-se que o ouvido humano tem maior sensibilidade ao som na faixa entre 1 kHz e 5 kHz. Todos os pontos da curva são percebidos com a mesma sonoridade (loudness), mas a pressão sonora necessária não é constante com a freqüência.

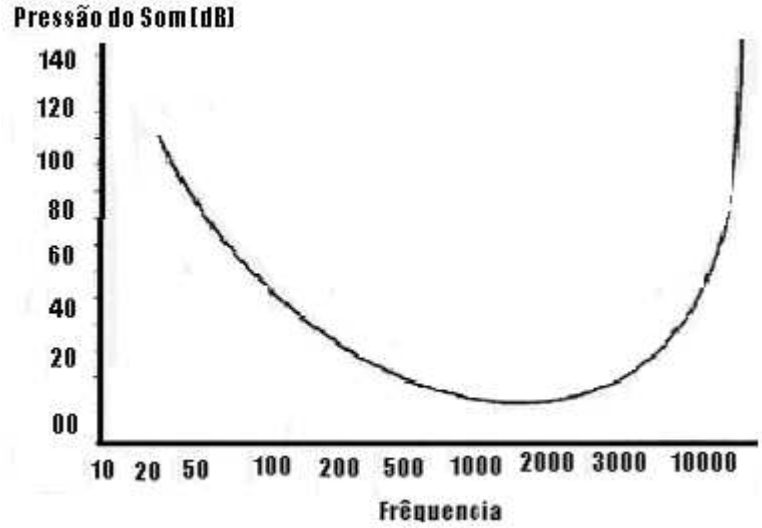

Figura 2.2: Sonoridade (loudness) percebida pelo ouvido humano.

A menor variação de pressão do ar que um ser humano pode detectar (20 micro Pascal) medido nas freqüências onde somos mais sensíveis, é usada como referência (0 dB) para medir a intensidade de outros sons.

$$Potência\ em\ dB(decibéis) = 10 \cdot \log \frac{P}{P_0} \tag{2.1}$$

onde P é a potência sendo considerada e P_0 é a potência correspondente a 20 micro Pascal. Uma conversa normal fica entre 50 e 60 dB e o som de carros em trânsito é aproximadamente de 80 dB. O máximo som que o ouvido pode tolerar é de 130 dB, o que dá um alcance dinâmico de 0 a 130 dB.

2.2 Mascaramento Auditivo

Um interessante fenômeno auditivo é o do mascaramento, onde um bloco de sons encobre a percepção de outra onda sonora. O mascaramento auditivo é definido como a "audibilidade diminuída de um som devido à presença de outro". O mascaramento auditivo consiste de mascaramento em freqüência e mascaramento temporal.

O mascaramento em freqüência também chamado mascaramento simultâneo é explicado melhor com um exemplo. Considere um som forte com uma freqüência de 1 kHz, e também um som na freqüência de 1.1 kHz que está a 18 dB abaixo do anterior. O som de 1.1 kHz não pode ser ouvido porque está sendo mascarado pelo som mais forte de 1 kHz. Isto ocorre porque o som de 1 kHz é mais forte e está perto em freqüência. Quanto maior a proximidade dos dois sinais no espectro de freqüência e quanto maior a diferença de potência, mais forte será o mascaramento (Figura 2.3).

O mascaramento temporal ocorre antes e depois de um som forte. Se um som é mascarado depois de um som mais forte é chamado de pós-mascaramento, e se é mascarado antes em tempo é chamado de pré-mascaramento. O pré-mascaramento existe só por um curto momento (20 ms). O pós-mascaramento tem efeito até por 200 ms (Figura 2.4).

Explorando ambos os mascaramentos, em freqüência e temporal, é possível reduzir substancialmente a informação de áudio sem uma mudança audível.

O fato de um som abafar outro parece até razoável. Porém, o efeito decorrente da inibição de um tom puro com ruído não é tão intuitivo. Neste caso, somente uma pequena extensão ao redor do tom é realmente abafada. Essa pequena extensão de freqüência é chamada de banda crítica.

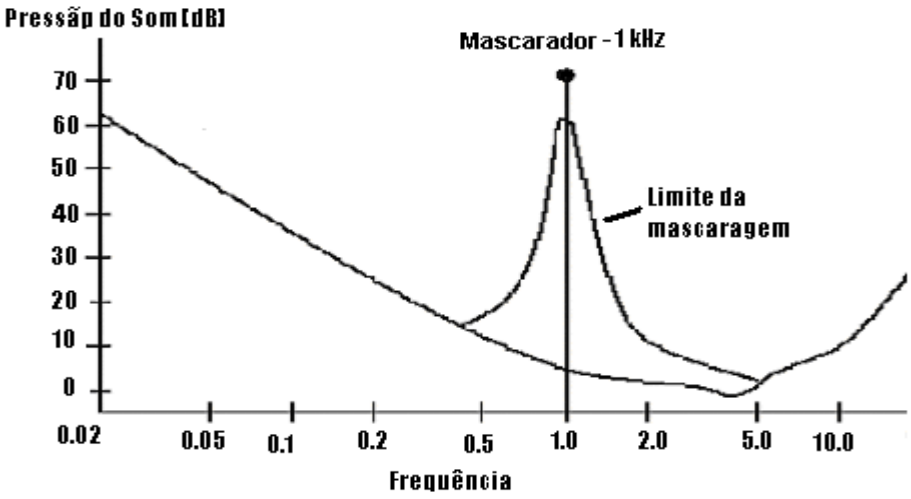

Figura 2.3 : Mascaramento em Freqüência

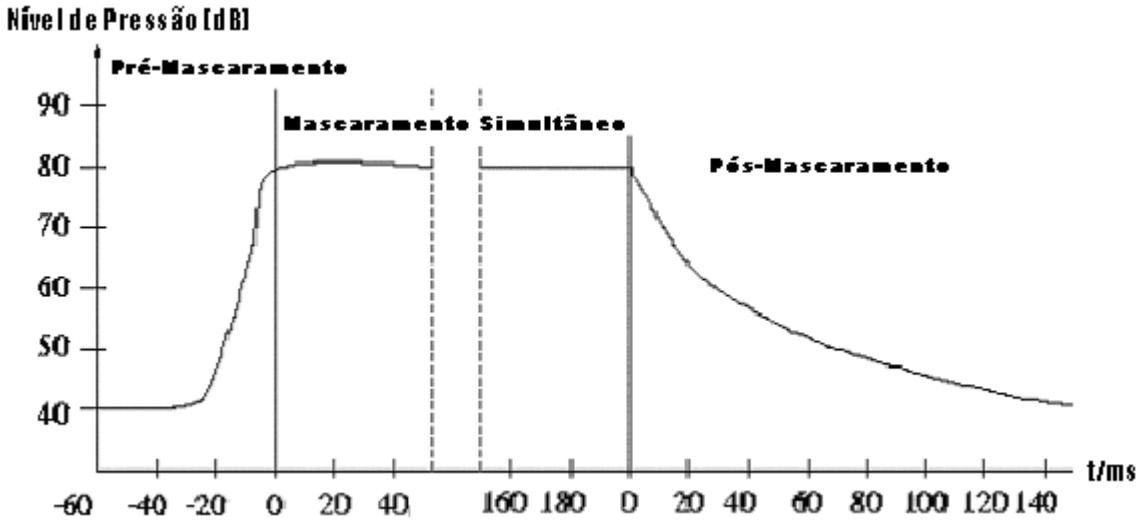

Figura 2.4 : Mascaramento temporal

3. Fundamentos

Este capítulo introduz os métodos de digitalização, os quais irão originar sinais modulados por codificação de pulsos que serão designados como sinais PCM (Pulse Code Modulation). A Figura 3.1 mostra os processos de digitalização.

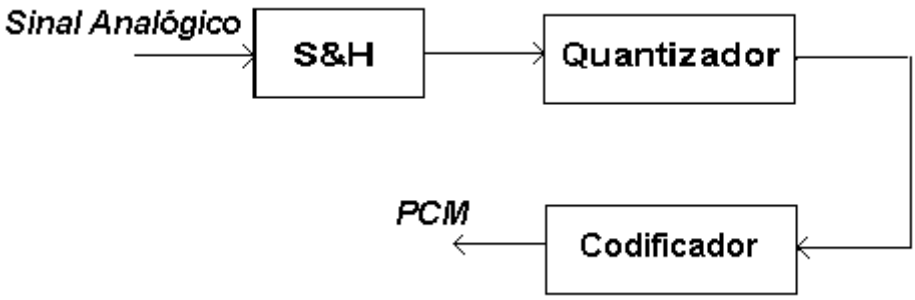

Figura 3.1 – Arquitetura de Digitalização

Nessa arquitetura destacam-se o bloco do amostrador com retenção (S&H - Sample e Hold) responsável pela amostragem temporal do sinal analógico, o quantizador que a cada instante de amostragem discretiza a amplitude do sinal e, por último, o codificador que codifica a amostra temporal quantizada (o código resultante é o binário).

3.1 Amostragem

A amostragem temporal é baseada no Teorema da Amostragem que afirma o seguinte: " Se o sinal for amostrado em intervalos regulares de tempo e a uma taxa duas vezes maior que a mais alta freqüência do sinal, então o sinal amostrado contém toda informação do sinal original".

Sendo assim, por exemplo no sistema telefônico, o sinal de voz utiliza uma banda de freqüência que se estende de 300 Hz até 3.4 kHz. A partir do Teorema da Amostragem, foi instituído que a freqüência de amostragem seria de 8 kHz, ou seja, mais do que duas vezes a maior freqüência do sinal de voz [4].

O processo de amostragem pode ser visualizado na Figura 3.2 .

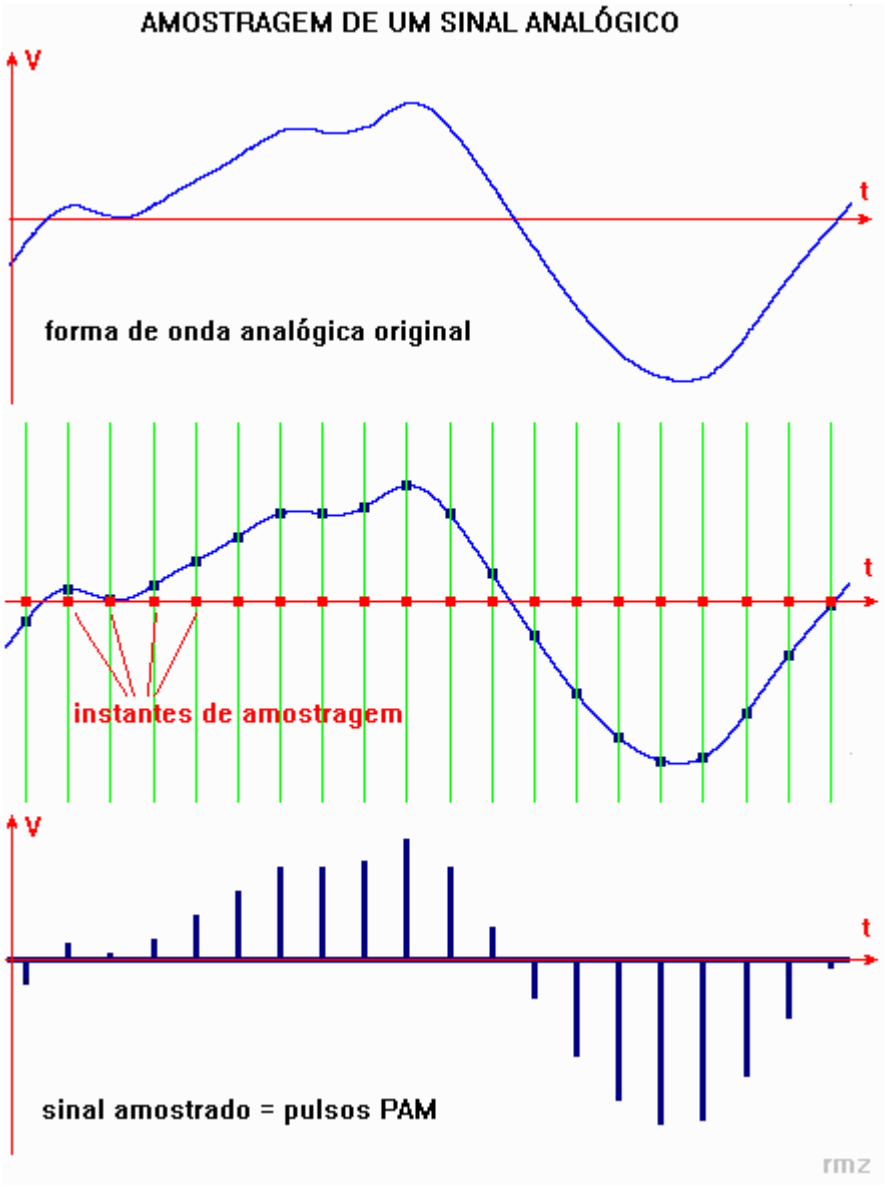

Figura 3.2 - Amostragem

A quantização ocorre após a discretização conduzida pela amostragem, que na prática recolhe a amostra no instante de amostragem e retém o respectivo valor da amplitude. Na etapa de quantização faz-se a correspondência entre o valor da amplitude colhida, que pode assumir uma infinidade de valores possíveis, em uma representação que possua um número finito de valores. Isso é possível através da divisão do eixo das amplitudes em um número finito de níveis. No caso de todos intervalos entre dois níveis possuírem o mesmo comprimento a quantização é dita uniforme.

Desta forma, amplitudes com valores diferentes poderão pertencer a um mesmo intervalo de quantização, isso por sua vez irá introduzir um ruído ao sinal, o ruído de quantização.

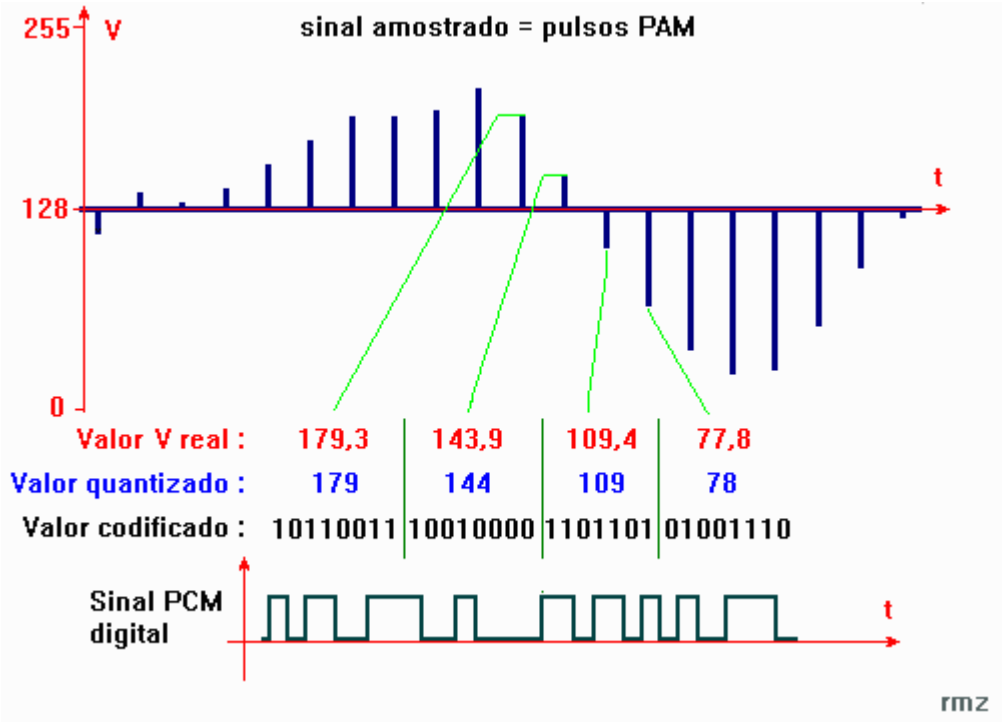

Figura 3.3 – Quantização e Codificação

Para concluir a digitalização procede-se a codificação, que nada mais é do que a transformação dos valores dos níveis em uma representação binária, ou seja, os valores decimais vinculados aos níveis de quantização serão codificados e representados em código binário, ou seja, por 0's e 1's . Tanto a quantização, quanto a codificação estão esquematizadas na Figura 3.3.

3.2 Quantização Não-Uniforme

Na quantização dita uniforme o intervalo dos níveis possuem o mesmo comprimento, fato que não ocorre na quantificação não-uniforme.

As amostras do sinal de voz ocorrem com uma densidade de probabilidade exponencial decrescente (Laplaciana), ou seja, ocorre um maior número de amostras com níveis de tensão mais baixos do que amostras com valores de tensão altos. Logo, aumentando o número de níveis de quantização na faixa de valores de tensão onde ocorrem mais amostras e diminuindo o número de níveis de quantização onde ocorrem menos amostras, é possível reduzir a potência do erro de quantização e, simultaneamente, manter o número total de níveis de quantização constante.

Observa-se também que com a quantização não uniforme, a relação entre a potência do sinal e a potência do ruído de quantização na saída do decodificador no receptor é praticamente constante com a variação da potência do sinal na entrada do codificador no transmissor.

4. Princípios de compressão

As técnicas de compressão freqüentemente se aproveitam da estrutura da representação dos dados para que utilizando certas características seja possível fazer uma representação mais compacta do sinal original. Normalmente essas técnicas levam em conta a redundância dos dados e/ou as propriedades da percepção humana.

A redundância de dados leva em consideração, entre muitas outras características, a correlação entre amostras, que é determinante do grau de semelhança existente entre amostras subseqüentes. Outro tipo de redundância é os momentos de silêncio num sinal de áudio.

Já as propriedades da percepção humana ocorrem devido ao fato de os seres humanos serem tolerantes a certos erros ou supressões de informação sem que isso comprometa a efetividade da comunicação. Sendo assim, pode-se representar a informação original de maneira diferente, sem que isso afete a comunicação.

No caso do áudio, pode-se dizer, fisicamente, que o som é o mecanismo no qual distúrbios propagam através do ar. Nessa definição física, não há limites para a freqüência ou os níveis a serem considerados. Porém, biologicamente, podemos afirmar que o ouvido humano somente responde a certas faixas de freqüência. Essa última seria mais bem adequada para definir os sons, ou seja, os sons são reproduções de uma faixa de freqüência e níveis que podem ser detectados pelo ouvido humano. Sendo assim, descreve-se o ouvido humano como tendo resolução finita em ambos domínios tempo e freqüência. Portanto, alguns distúrbios são inaudíveis para o ser humano podendo ser suprimidos.

Por definição, o som de qualidade de um codificador só pode ser determinado pelo ouvido humano. Igualmente, um poderoso codificador pode somente ser projetado com um bom conhecimento dos mecanismos do ouvido humano.

Os métodos de compressão de áudio são divididos em dois grupos, a compressão com perda de informação e a compressão sem perda. Quando falamos em técnicas de compressão ou algoritmo de compressão, dois algoritmos são referenciados. Há o algoritmo de compressão que obtém uma entrada de dados X e gera uma representação X_C idêntica a entrada, ou então, um algoritmo que a partir de uma entrada X e gera uma representação X_C que difere da entrada, esse é o caso de compressão com perdas.

4.1 Compressão sem Perdas (Lossless Compression)

Na compressão sem perdas, como o próprio nome diz, não há perda de informação, ou seja o sinal original passa pelo processo de compressão e descompressão e é recuperado exatamente igual ao sinal original. A Compressão sem perdas é utilizado em aplicações que não toleram diferenças entre o sinal original e o reconstruído.

Um exemplo dessa situação acontece quando suponha que a imagem de um raio-x seja comprimida e transmitida para ser diagnosticado por um médico , se por algum motivo o sinal original for diferente do sinal reconstruído, o medico pode ser levado a errar e dessa forma, pode-se colocar em perigo uma vida humana. Porém em outras aplicações há tolerâncias a erros, nessa situação é usadas a compressão com perda que será apresentada no Capitulo 4.2.

4.2 Compressão com Perdas (Lossy Compression)

Esse tipo de compressão envolve algum tipo de perda de informação, ou seja, o sinal recuperado não é exatamente igual ao sinal original. Porém a vantagem que a tolerância às distorções traz é a possibilidade de se obter maiores taxa de compressão do que numa compressão sem perdas.

Em muitas aplicações a recuperação de um sinal diferente não traz danos. Por exemplo, pode-se citar o sistema telefônico, que dependendo da qualidade requerida pode tolerar perdas de informação sem contudo comprometer a sua função. Contudo, em algumas situações, a perda de informação é tolerada quando a quantidade é pequena, como é o caso Compact Disc (CD) de alta qualidade.

4.3 Desempenho

Uma maneira lógica de se saber o quão bom é um algoritmo de compressão é através da taxa de bits requerida para representar os dados após a compressão A relação entre a quantidade de bits requerida antes e depois da compressão é chamada de fator de compressão. Outra maneira de se quantificar é sabendo-se o número médio de bits necessário para representar uma única amostra.

Em contrapartida, em compressões com perdas é preciso determinar de alguma forma a eficiência do algoritmo de compressão, já que o sinal reconstruído difere do original. A partir dessa discussão pode-se tentar responder a seguinte pergunta:

Como medir a fidelidade de uma seqüência reconstruída da original?

A resposta freqüentemente depende do que está sendo comprimido e de quem está fazendo a pergunta. Se por acaso a compressão for de uma imagem, uma obra de arte, o melhor meio para descobrir o quanto de distorção foi introduzido é perguntando para um membro da família proprietária da obra a sua opinião sobre a imagem.

Contudo se a imagem é de um satélite e foi processada por uma máquina, a melhor medida da fidelidade é ver como a distorção introduzida afeta o funcionamento da máquina. Similarmente, se a compressão é então a reconstrução de um segmento de áudio, o julgamento sobre a fidelidade entre a seqüência da reconstrução e o original depende do tipo de material que está sendo examinado assim como do contexto no qual o julgamento é feito. Por exemplo, a distorção é muito mais fácil de ser notada em um pedaço de música do que numa conversa policial.

Dessa forma, conclui-se que o ideal seria usar o usuário final da saída de uma fonte particular para determinar a qualidade e fornecer a realimentação requerida pelo projeto. Na prática isto não é possível, especialmente quando o usuário final é um ser humano, pois a resposta humana é difícil de ser representada matematicamente.

Uma outra alternativa, menos subjetiva, é a observação da diferença entre a seqüência original e os valores reconstruídos. Duas populares medidas de distorção ou diferença entre o original e a seqüência reconstruída são a medida do quadrado do erro e a medida do erro absoluto.

Portanto se $\{X_n\}$ é a saída da fonte e $\{Y_n\}$ é a seqüência reconstruída, então a medida do quadrado do erro $d(X,Y)$ é dado por:

$$d(X,Y) = (X - Y)^2 \tag{4.1}$$

E a medida do erro absoluto $d_a(X,Y)$ é:

$$d_a(X,Y) = |X - Y| \tag{4.2}$$

Em geral é complicado examinar termo a termo. Por isso, é usual utilizar um número médio para facilitar a observações sobre a diferença entre as seqüências $\{X_n\}$ e $\{Y_n\}$.

Freqüentemente, é utilizado a medida do erro quadrático médio, que é representado pelo símbolo σ_d^2, onde:

$$\sigma_d^2 = \frac{1}{N} \cdot \sum_{n=1}^{N} (X_n - Y_n)^2 \tag{4.3}$$

Se o interesse está no tamanho relativo do erro do sinal, pode-se encontrar a razão entre o valor da variância da saída de uma fonte σ_X^2 e o erro quadrático médio.

Isso é chamado de razão sinal–ruído (*SNR*) :

$$SNR = \frac{\sigma_X^2}{\sigma_d^2} \tag{4.4}$$

O SNR é freqüentemente utilizado em sua forma na escala logarítmica e representado em decibéis (dB):

$$SNR = 10 \cdot \log_{10} \frac{\sigma_X^2}{\sigma_d^2} \tag{4.5}$$

Sendo assim, nesse item observamos dois métodos para medir a fidelidade da reconstrução. O primeiro método envolve seres humanos podendo fornecer uma grande medida de acurácia , porém esse método não é prático e não é usado em projetos de aproximação matemática.

O segundo é matematicamente tratável, mas não fornece uma grande indicação de acurácia. Um meio termo seria encontrar modelos matemáticos para percepção humana. Há dois problemas com esta aproximação, o primeiro seria ao fato do processo de percepção humana ser muito difícil de modelar. O segundo problema é o fato de que em um modelo matemático para percepção humana, a adição seria tão complexa que poderia ficar matematicamente intratável.

4.4 Teoria da Informação

Shannon propôs a idéia de se medir a quantidade de informação o que fez surgir a teoria da informação. Suponha que exista um evento A, o qual é um conjunto de resultado de algum experimento. Se P(A) é a probabilidade do evento A ocorrer, então a quantidade de informação $i(A)$ associada é dada por:

$$i(A) = \log_b \frac{1}{P(A)} = -\log_b P(A) \tag{4.6}$$

O uso do logarítmo para se obter a medida da informação não foi uma escolha arbitrária. Note que se a probabilidade de um evento é pequena, a quantidade de informação associada a ele é alta, se a probabilidade é grande, então a quantidade de informação associada é baixa. Se for usada a base como sendo igual a 2, então a unidade de informação é dada em *bits*.

Por outro lado se há um conjunto de um evento independente A_i , tal que um conjunto de resultados seja S, então:

$$\bigcup A_i = S \tag{4.7}$$

onde S é o espaço de amostras, então a média da quantidade de informação associada com esse experimento aleatório é dada por:

$$\mathrm{H} = \sum P(A_i) i(A_i) = -\sum P(A_i) \log_b P(A_i) \tag{4.8}$$

Esta quantidade H é denominada entropia. Shannon mostrou que se o experimento origina os símbolos A_i de um conjunto A, então a entropia será a medida do número médio de símbolos binário necessário para codificar o sinal de saída.

Por exemplo, considere o conjunto A = {1,2,..,m} gerando uma seqüência de elementos $\{X_1, X_2,...\}$, que são independentes e igualmente distribuídos, então a relação que define a entropia torna-se :

$$\mathrm{H} = -\sum P(X_1) \log_b P(X_1) \tag{4.9}$$

5. Modelos para compressão sem perdas

É nessa fase que se tenta extrair informações sobre qualquer redundância que possa existir nos dados e a partir disso descrevê-la na forma de um modelo. Um modelo pode ser vantajoso na estimativa da entropia. Além disso, bons modelos são garantia de um eficiente algoritmo de compressão.

A manipulação dos dados utilizando operações matemáticas requer antes de tudo um modelo matemático. Se for conhecido algum aspecto físico sobre a geração dos dados, pode-se utilizar essa informação para construir um modelo, dessa forma estará se construindo um modelo físico. Por exemplo, em aplicações de processamento de sinais de voz o conhecimento físico de como o sinal de voz é produzido pode ser usado na construção de um modelo matemático para o processo de amostragem do sinal de voz.

Considerando um modelo estatístico deve-se assumir que cada elemento gerado por um determinado evento em questão, é independente de um elemento gerado por outro evento. Desta forma, através de um modelo probabilístico, obtém-se o valor da entropia através da Eq. 4.9. Este parâmetro será utilizado na construção de um código eficiente.

Por último, pode-se representar a dependência dos dados através do uso do Modelo de Markov. Em compressões sem perdas usa-se um tipo específico desse modelo.

Supondo uma seqüência de observação $\{X_n\}$, que é dada pelo seguinte modelo de Markov.

$$P(X_n \mid X_{n-1},..., X_{n-k}) = P(X_n \mid X_{n-1},..., X_{n-k},...) \tag{5.1}$$

O conhecimento do passado de “k” símbolos é equivalente ao conhecimento de todo o processo. Os valores do conjunto $X_{n-1}, X_{n-2},..., X_{n-k}$ é chamado de estados do processo. Se a quantidade de símbolos no evento é *l*, então l^k é o numero de estados. O modelo mais comum é o modelo de Markov de 1º ordem.

$$P(X_n \mid X_{n-1}) = P(X_n \mid X_{n-1}, X_{n-2}, X_{n-3},...) \tag{5.2}$$

As equações 5.1 e 5.2 indicam que há dependência entre as amostras. O desenvolvimento do modelo de Markov de 1º ordem depende das premissas em relação à dependência entre as amostras.

5.1 Codificação de Huffman

Os procedimentos da codificação de Huffman são baseados na seguinte observação:

- Em um código otimizado, símbolos que acorrem mais freqüentemente terão um menor código resultante que os símbolos que ocorrem menos freqüentemente.

Na primeira observação, se os símbolos que ocorrem mais freqüentemente possuem um código resultante maior do que o código resultante de símbolos que ocorrem menos, então o número médio de bits requeridos seria maior do que na situação inversa.

Para se construir um código de Huffman é preciso primeiramente reconhecer a probabilidade de ocorrência dos símbolos. Supondo o seguinte exemplo [1]:

Tabela 5.1: Probabilidade de Ocorrência dos Símbolos

Símbolo (A_i)	Probabilidade $P(A_i)$	Código Resultante
A_2	0,4	$C(A_2)$
A_1	0,2	$C(A_1)$
A_3	0,2	$C(A_3)$
A_4	0,1	$C(A_4)$
A_5	0,1	$C(A_5)$

Inicialmente pode-se observar a entropia dessa fonte como sendo $2{,}122\ \text{bits/símbolo}$. Também se visualiza que os dois símbolos com menor probabilidade são A_4 e A_5. Portanto o código para esses símbolos será:

$$C(A_4) = \alpha_1 * 0$$

$$C(A_5) = \alpha_1 * 1$$

onde α_1 é uma string binária e * significa concatenação. Agora nós iremos redefinir os símbolos e suas respectivas probabilidades:

Tabela 5.2: Primeira Redefinição de Símbolos

Símbolo (A_i)	Probabilidade P(A_i)	Código Resultante
A_2	0,4	$C(A_2)$
A_1	0,2	$C(A_1)$
A_3	0,2	$C(A_3)$
A_4'	0,2	α_1

Como se pode visualizar :

$$P(A_4') = P(A_4) + P(A_5) = 0{,}2$$

Seguindo, escolhe-se os dois símbolos com menor probabilidade, nesse caso A_4' e A_3 . Sendo assim:

$$C(A_3) = \alpha_2 * 0$$

$$C(A_4') = \alpha_2 * 1 \quad e \quad C(A_4') = \alpha_1 \quad \therefore \quad \alpha_1 = \alpha_2 * 1$$

Isto significa o seguinte:

$$C(A_4) = \alpha_2 * 10$$

$$C(A_5) = \alpha_2 * 11$$

Redefinindo os símbolos:

Tabela 5.3: Segunda Redefinição de Símbolos

Símbolo (A_i)	Probabilidade P(A_i)	Código Resultante
A_2	0,4	$C(A_2)$
A_3'	0,4	α_2
A_1	0,2	$C(A_1)$

Novamente deve-se identificar os dois símbolos com menor ocorrência (A_1 e A_3').

$$C(A_3') = \alpha_3 * 0 \quad e \quad C(A_3') = \alpha_2 \quad \therefore \quad \alpha_2 = \alpha_3 * 0$$

$$C(A_1) = \alpha_3 * 1$$

O que significa:

$$C(A_3) = \alpha_2 * 0 = \alpha_3 * 00$$

$$C(A_4) = \alpha_2 * 10 = \alpha_3 * 010$$

$$C(A_5) = \alpha_2 * 11 = \alpha_3 * 011$$

Como restaram apenas dois símbolos procede-se diretamente

$$C(A_3') = 0 \quad e \quad C(A_3') = \alpha_3 * 0 \quad \therefore \quad \alpha_3 = 0$$

$$C(A_2) = 1$$

O resultado final, então, é o seguinte:

Tabela 5.4: Resultado final

Símbolo (A_i)	Probabilidade $P(A_i)$	Código Resultante
A_2	0,4	1
A_1	0,2	001
A_3	0,2	000
A_4	0,1	0010
A_5	0,1	0011

A média de bits/símbolos do código é:

$$l = 0{,}4 \cdot 1 + 0{,}2 \cdot 2 + 0{,}2 \cdot 3 + 0{,}1 \cdot 4 + 0{,}1 \cdot 4 = 2{,}2 \ \ bits/símbolo$$

A redundância pode ser encontrada através da seguinte equação

$$R = l - H \tag{5.3}$$

Para este exemplo particular, a entropia pode ser calculada usando a Eq. 4.9:

$$\mathrm{H} = \sum_{i=1}^{5} P(A_i) \log_2 P(A_i) = 2{,}122 \ bits/símbolo$$

Portanto, a redundância será de $0{,}078 \ bits/símbolo$.

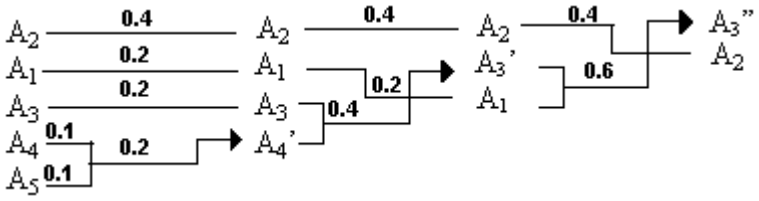

Figura 5.1 – Árvore binária

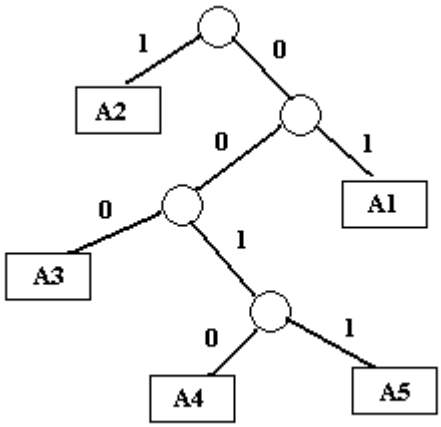

Figura 5.2 – Outra Forma de Representação da Árvore binária

5.1.1 Codificação Adaptativa de Huffman

A codificação de Huffman requer o conhecimento da probabilidade dos símbolos gerados pela fonte. Porém se isso não é atendido, passa a ser necessário a introdução de um procedimento de codificação em dois passos: coletar a estatística de ocorrência da fonte e codificá-la.

Porém é possível converter esses dois procedimentos em apenas um passo. Para isso foram desenvolvidos alguns algoritmos adaptativos, no intuito de se codificar baseado na estatística de símbolos já colhidos.

Teoricamente, pretende-se codificar *(k+1)* símbolos usando a estatística dos primeiros *k* símbolos. Sendo assim, pode-se recalcular o código com a codificação de Huffman cada vez que um símbolo é transmitido. O código de Huffman pode ser descrito em termos de uma árvore binária. O código resultante para um símbolo pode ser obtido através da análise da árvore começando na raiz e indo até as folhas, onde 0 corresponde aos galhos da esquerda e 1 aos da direita. Também é necessário ter o conhecimento de dois parâmetros: o peso de cada folha e o número do nó.

O peso do nó externo é simplesmente o número de vezes que o correspondente símbolo foi encontrado, enquanto o peso do nó interno constitui a soma dos pesos de seus antecessores. O número do nó refere-se ao número relacionado aquela folha.

Durante a codificação adaptativa tanto o transmissor quanto o receptor possuem algum conhecimento estatístico da fonte. A árvore de ambos (receptor e transmissor) consiste de um simples nó que corresponde a todos símbolos *ainda não transmitidos* (NYT – Not Yet Transmitted) e possui peso igual a zero. Com o progresso da transmissão os nós relacionados aos símbolos transmitidos serão adicionados na árvore e a árvore irá sendo configurada repetidas vezes a partir de um procedimento de *update*. Antes do inicio da transmissão, receptor e transmissor definem códigos fixos para cada símbolo.

Quando um símbolo é transmitido pela primeira fez, o código para nó NYT é transmitido, seguido pelo código fixo do símbolo. O nó para o símbolo é criado na árvore binária e retirado da lista de NYT. Transmissor e receptor iniciam com a mesma estrutura de árvore e o procedimento de *update* (atualização) mantém, o que faz esse processo síncrono.

5.1.2 Procedimento de Update

Esse procedimento mantém os nós numa ordem fixa que é baseada num critério probabilístico, ou seja, há um ordenamento onde os símbolos que ocorre mais freqüentemente posicionam-se na raiz da árvore e os de menor freqüência no final. Isso é feito da seguinte forma: o maior número do nó é dado pela raiz da árvore e o menor pelo nó do NYT. O número do nó NYT é especificado em ordem crescente da esquerda para a direita e a partir do maior nível. Conjuntos de nós com o mesmo peso formam um bloco. Devido aos procedimentos de *update*, ambos, receptor e transmissor, operam com a mesma informação, isso possui a função de preservar a propriedade de semelhança. O *update* é feito no receptor logo após a decodificação e no transmissor após a codificação.

O procedimento flui da seguinte forma: o nó externo é analisado para se verificar se ele possui o maior número. Caso ele não o seja, troca-se ele com o nó do bloco com o maior número. A atualização é processada assim que o nó com o maior número é encontrado.

O peso do nó externo é incrementado e passa-se a verificar se o antecessor do nó é agora o de maior número. Se não, troca-se ele com o nó de maior número no bloco. Da mesma forma, incrementa-se o peso desse nó. Esse processo é repetido até a raiz da árvore ser analisada.

Se o símbolo a ser processado ocorreu pela primeira vez, um novo nó externo é criado e referenciado a esse símbolo. Um novo nó NYT é adicionado no final da árvore. Ambos, o novo nó externo e o novo

NYT são antecedidos pelo antigo NYT. Em seguida incrementa-se o peso do novo nó externo e do antigo NYT. Da mesma forma repete-se o procedimento até chegar a raiz da árvore.

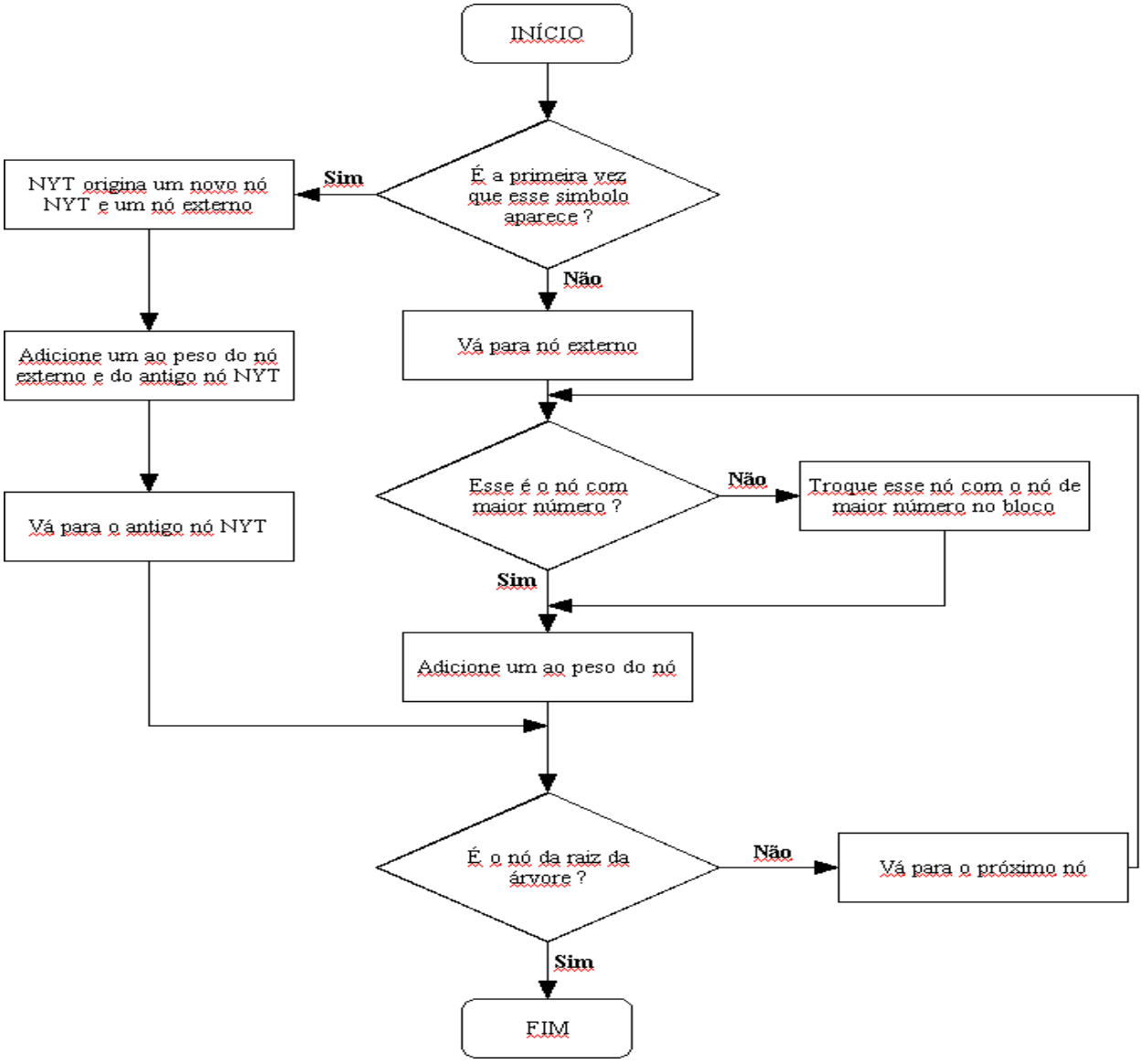

Figura 5.3 – Fluxograma do Update

5.1.3 Procedimentos de Codificação

Na codificação, as árvores binárias do transmissor e do receptor consistem de um único nó, o NYT, onde consta a lista dos códigos de todos os símbolos que possam ser gerados pela fonte.

Quando o primeiro símbolo é transmitido, é enviado o código do nó NYT seguido pelo código do símbolo anteriormente definido. Sendo assim, esse código é retirado da lista de símbolos ainda não transmitidos (NYT) e a atualização então é processada. No caso de um símbolo já transmitido ser gerado novamente pela fonte de informação, é enviado apenas o código da posição do símbolo na

árvore binária. Como a ordem da árvore faz com que o posicionamento dos símbolos de maior ocorrência seja na raiz e os de menor freqüência no final, os códigos que tiverem um histórico de transmissão maior, obterão uma palavra codificada com menos bits.

A fonte de informação gera um alfabeto $\{A_1,A_2,...,A_m\}$ com m símbolos tal que $m = 2^e + r$ e $0 \le r \le 2^e$. Sendo assim, o símbolo A_k será codificado como um palavra binária de $(e + 1)$ bits representando o número decimal $(k - 1)$, se $1 \le k \le 2r$. Senão, A_k será codificado como uma palavra de e bits que representará o número decimal $(k - r - 1)$. O fluxograma da codificação se encontra na Figura 5.4.

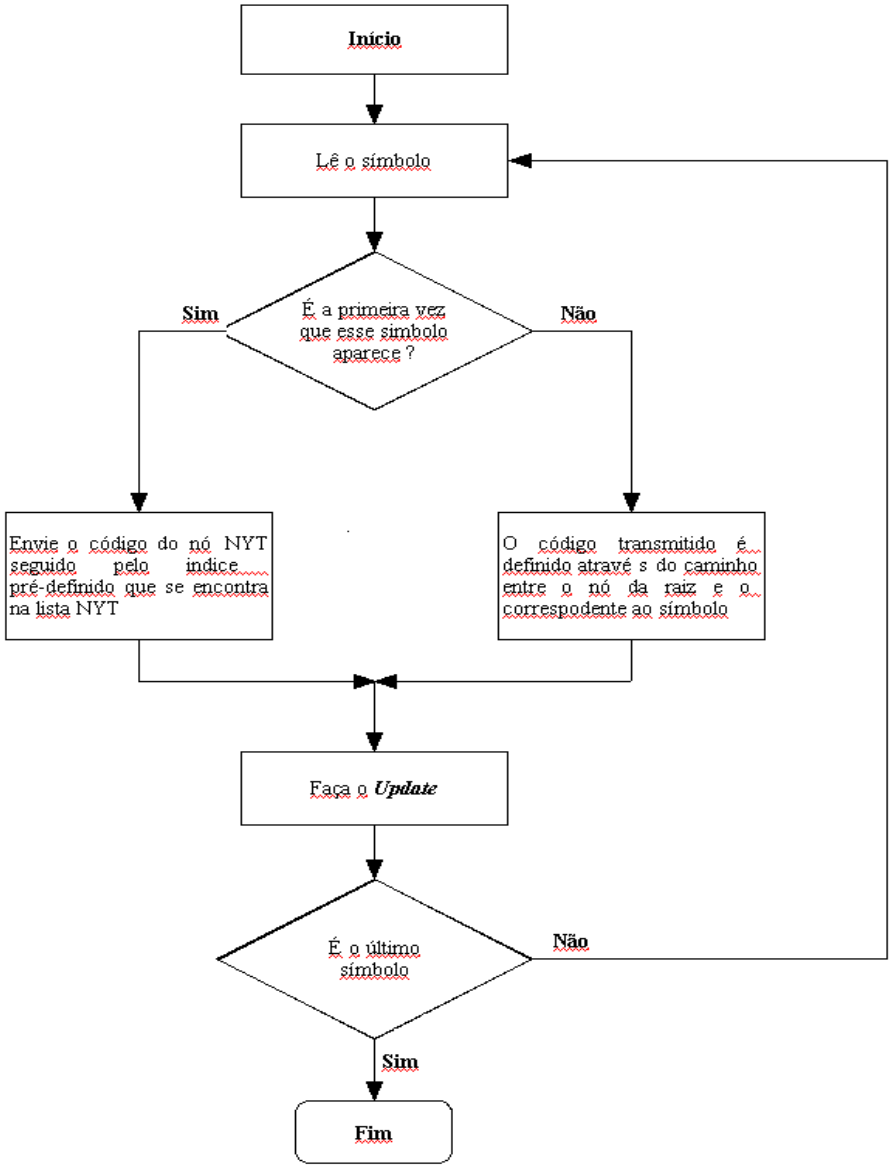

Figura 5.4 – Fluxograma da Codificação

5.1.4 Procedimento de Decodificação

O procedimento de codificação é feito lendo-se a árvore binária da raiz até as folhas, quando se encontra a folha referente ao código recebido, o símbolo é então decodificado. Se o símbolo está sendo enviado pela primeira vez, checa-se os ***e*** bits que se encontram após o código do nó NYT. No caso do numero binário convertido para decimal ser menor do que ***r***, é lido o próximo bit. Logo que o símbolo é decodificado, o procedimento de *update* inicia-se e em seguida passa-se ao próximo código recebido. O fluxograma da decodificação se encontra na Figura 5.5.

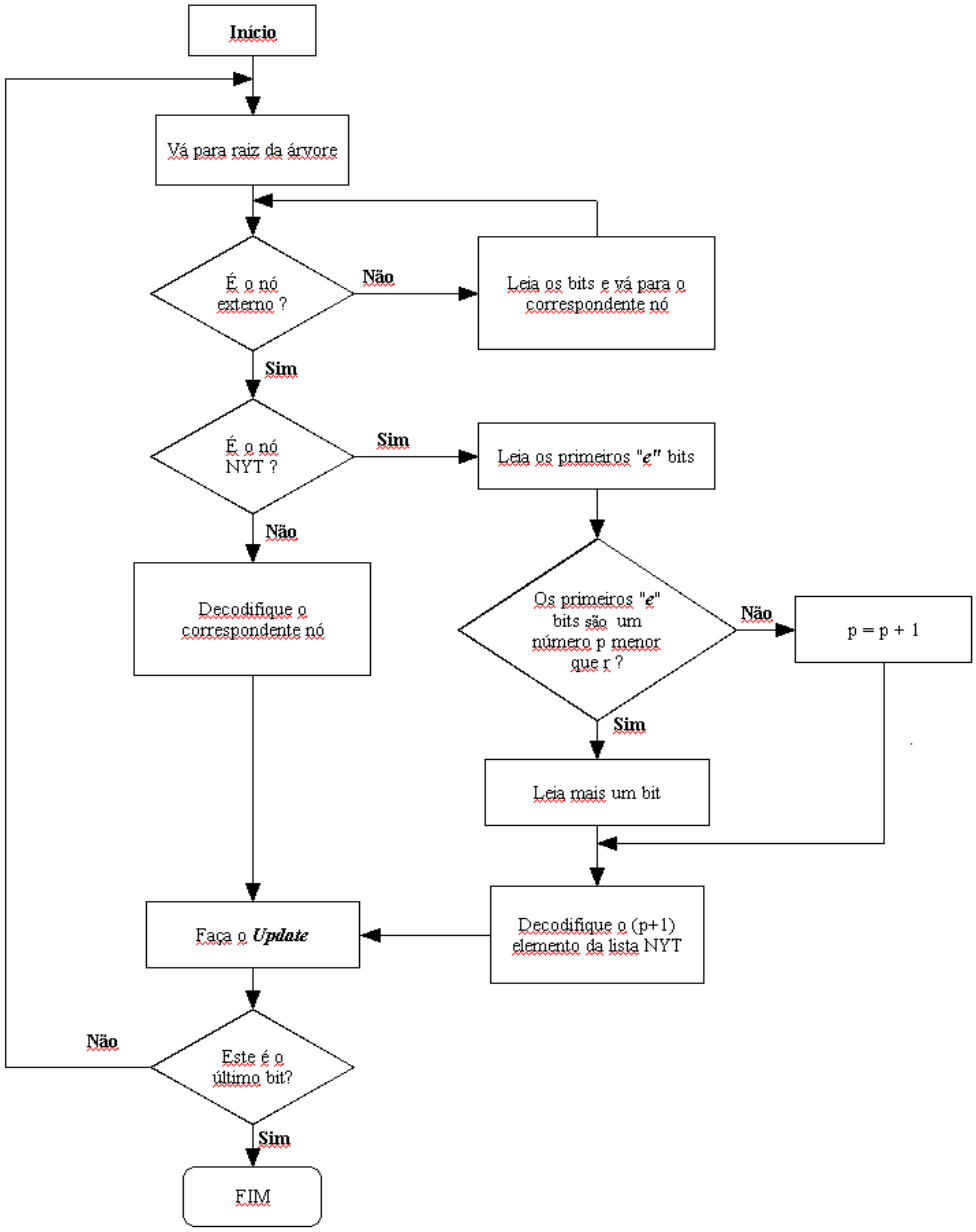

Figura 5.5 – Fluxograma da Decodificação

6. Modelos de Compressão com Perdas

O conjunto de modelos para compressões com perdas é muito maior do que o exposto para compressões sem perdas. Não serão apresentados todos por isso significar uma exaustiva lista de modelos. Contudo, pretende-se descrever alguns desses modelos.

6.1 Modelo Probabilístico

Um importante método para caracterizar uma fonte é através do uso de modelos probabilísticos. Como será observado, o conhecimento desse tipo de modelo é importante no projeto de um número de esquemas de compressão.

Quando se desenvolvem modelos tenta-se uma matemática exata. A probabilidade de cada símbolo é estimada com parte do processo de modelamento. Certas funções de distribuição probabilísticas são mais analiticamente tratáveis do que outras, sendo que o objetivo aqui é escolher a melhor função de distribuição.

Distribuição Uniforme, distribuição Gaussiana e distribuição Laplaciana são três modelos comumente usados em projetos e análises de sistema de compressão com perdas.

- ***Distribuição Uniforme:*** No caso de não termos nenhum conhecimento sobre a distribuição da saída da fonte, pode-se usar esse tipo de distribuição. A função de densidade probabilística para uma variável aleatória uniformemente distribuída entre *a* e *b* é:

$$F(x) = \begin{cases} \dfrac{1}{(b-a)} & \text{para} \quad a < x < b \\ 0 & \text{caso contrário} \end{cases} \tag{6.1}$$

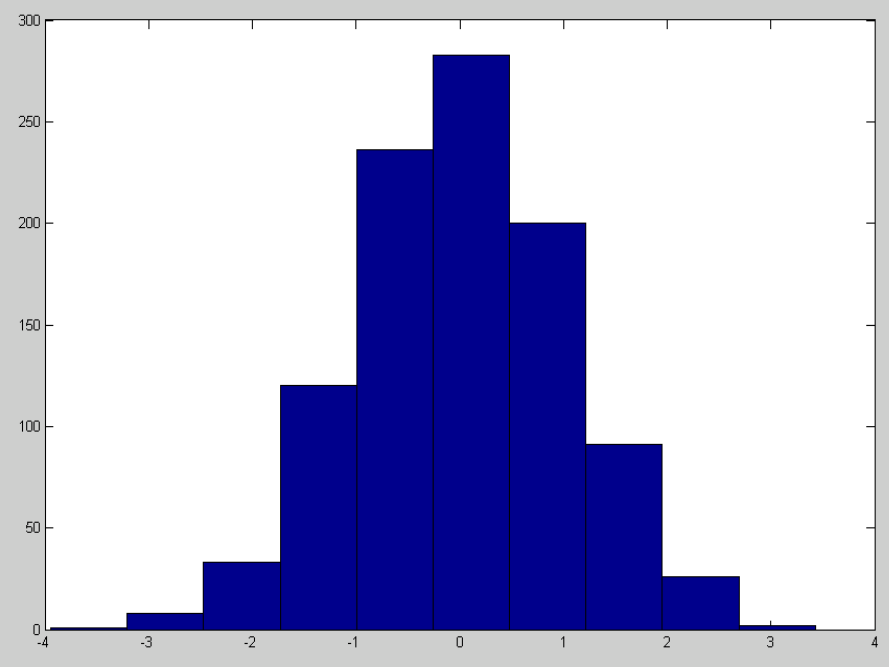

Figura 6.1 – Gráfico da Distribuição Gaussiana

- ***Distribuição Gaussiana:*** É uma das mais usadas por ser matematicamente tratável. A função de densidade de probabilidade para uma variável aleatória *x* de média μ e variância σ^2 é:

$$F(x) = \frac{1}{\sqrt{2 \cdot \pi \cdot \sigma^2}} e^{-\frac{(x-\mu)^2}{2 \cdot \sigma^2}} \tag{6.2}$$

- ***Distribuição Laplaciana:*** Várias fontes possuem distribuições com valores nulos. Por exemplo, na voz existem momentos de silêncio, ou seja, as amostras da voz com amplitude zero terão uma alta probabilidade de acontecer. Nessa situações, uma Distribuição Gaussiana não é recomendada. A distribuição comumente usada é a Laplaciana com média zero e variância σ^2 para uma variável aleatória *x*.

$$F(x) = \frac{1}{\sqrt{2 \cdot \sigma^2}} e^{\frac{-\sqrt{2}|x|}{\sigma}} \tag{6.3}$$

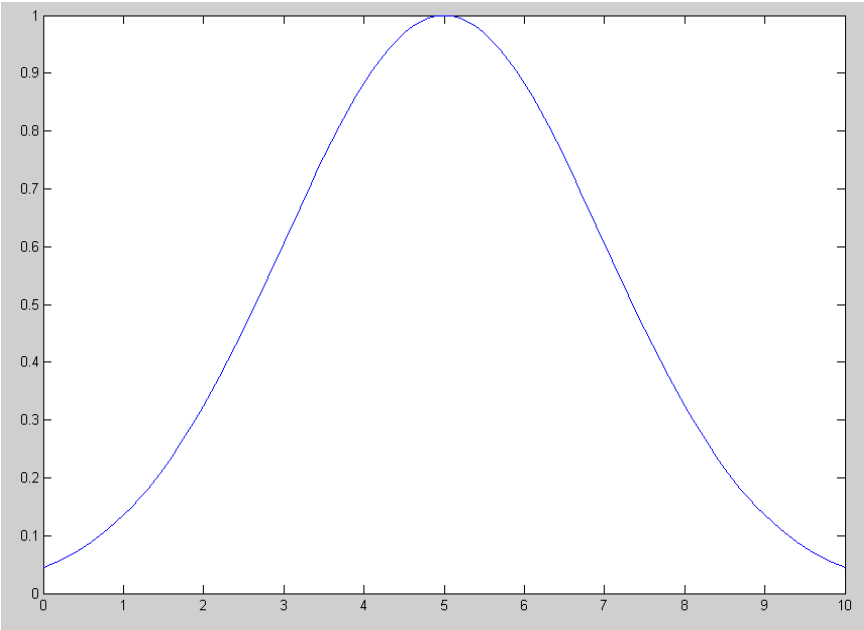

Figura 6.2 – Gráfico da Distribuição Gaussiana

6.2 *Modelos de Sistemas Lineares*

Uma grande classe de processos pode ser modelada na forma da seguinte equação de diferenças;

$$x_n = \sum_{i=1}^{N} a_i \cdot x_{n-i} + \sum_{j=1}^{M} b_i \cdot \varepsilon_{n-i} + \varepsilon_n \tag{6.4}$$

onde $\{x_n\}$ são amostras do processo que se deseja modelar e $\{\varepsilon_n\}$ é a seqüência do ruído branco.

Relembrando que a seqüência $\{\varepsilon_n\}$ possui uma função de auto-correlação:

$$R_{xx}(k) = \begin{cases} \sigma_x^2 & \text{para} \quad k = 0 \\ 0 & \text{caso contrário} \end{cases} \tag{6.5}$$

$$\textbf{Onde :}\ R_{xx}(k) = \begin{cases} \sigma_x^2 & \text{para} \quad k = 0 \\ 0 & \text{caso contrário} \end{cases} \tag{6.6}$$

Na terminologia de processamento digital de sinais, a Eq. 6.4 representa a saída de um filtro linear discreto e invariante no tempo com N pólo e M zeros. Na literatura estatística, esse modelo se chama Média do Movimento Autoregressivo de ordem (N,M), ou Modelo ARMA (N,M). O termo autoregressivo é devido ao primeiro somatório, e o segundo somatório denomina a outra porção do nome.

Se todos os b_j forem iguais a zero, restará apenas a parte autoregressiva. Esse modelo é chamado de AR(N). Ele é muito popular entre os modelos lineares, especialmente na compressão da voz. Primeiro, note que para um processo AR(N), conhecendo o passado do passado e não sabendo mais do que as últimas N amostras do processo, temos que:

$$P(X_n | X_{n-1}, X_{n-2}, ...) = P(X_n | X_{n-1}, X_{n-2}, ..., X_{n-N}) \tag{6.7}$$

Isto significa que o processo AR(N) é um modelo de Markov de ordem N. A função de auto-correlação pode dar maiores informações sobre o comportamento da seqüência de amostras. O decaimento lento da função de auto-correlação indica uma alta correlação de uma amostra para outra, enquanto um decaimento rápido denota o contrário. A função de auto-correlação para o processo AR(N) é obtida como a seguir:

$$R_{xx}(k) = E[x_n x_{n-k}] \tag{6.8}$$

$$R_{xx}(k) = E\left[\left(\sum_{i=1}^{N} a_i \cdot x_{n-i} + \varepsilon_n\right)(x_{n-k})\right] \tag{6.9}$$

$$R_{xx}(k) = E\left[\sum_{i=1}^{N} a_i \cdot x_{n-i} \cdot x_{n-k}\right] + E[\varepsilon_n \cdot x_{n-k}] \tag{6.10}$$

$$R_{xx}(k) = \begin{cases} \sum_{i=1}^{N} a_i \cdot R_{xx}(k-i) & \text{para } k > 0 \\ \sum_{i=1}^{N} a_i \cdot R_{xx}(i) + \sigma_\varepsilon^2 & \text{para } k = 0 \end{cases} \tag{6.11}$$

6.3 Modelos Físicos

Modelos Físicos são baseados na maneira como é produzida a saída da fonte. Geralmente, são modelos complicados e não são responsáveis por aproximações matemáticas razoáveis, com exceção para a geração da voz.

A voz é produzida pela força do ar que é expelida dos pulmões e é conduzida através do trato vocal que funciona como um filtro. A resposta em freqüência desse filtro é modificada a partir da forma do trato focal.

A forma como o trato vocal varia é lento, fazendo com que a função de transferência do filtro necessite ser atualizado a cada período de 20 ms. O som é criado pelas vibrações das cordas vocais

localizadas no trato vocal, que interrompendo a passagem do ar produz pulsos de ar como excitação, o que irá definir o tom de voz.

É nesse cenário que os codificadores se inserem para explorar as características do sinal de voz no intuito de reduzir a taxa de bits.

6.4 Codificadores de Forma de Onda

Os codificadores de forma de onda processam o sinal de entrada sem reconhecer o modo como esse sinal foi gerado, ou seja, ele produz o sinal reconstruído no intuito que ele se aproxime ao máximo do sinal original. Observando esses tipos de codificadores verificamos que eles funcionam independentemente do tipo de sinal.

6.4.1 Codificação DPCM

Fontes, tanto para voz como para imagens, possuem uma grande correlação entre amostras subseqüentes. Pode-se usar esse fato para predizer cada amostra baseada em seu passado e somente codificar e transmitir a diferença entre a predição e o valor verdadeiro da amostra.

Quando projetamos um quantizador para uma dada fonte, o tamanho do intervalo de quantização depende da variância da entrada.Se assumirmos que a entrada é uniformemente distribuída, a variância depende da extensão dinâmica da seqüência da entrada. O tamanho do intervalo de quantização determinará o ruído de quantização inserido durante o processo.

Em muitas fontes, as amostras geradas não se diferem uma da outra, isto significa que ambos, extensão dinâmica e a variância da seqüência da diferença $d_n = x_n - x_{n-1}$ são consideravelmente menor que se transmitíssemos a seqüência de saída.

Considerando uma seqüência $\{x_n\}$ e a seqüência diferencial $\{d_n\}$, que é gerada pelas diferenças $\{x_n - x_{n-1}\}$, pode-se proceder a quantização e obter $\{{}^*d_n\}$:

$$ {}^*d_n = Q[d_n] = d_n + q_n \tag{6.12}$$

onde $\{q_n\}$ é o erro de quantização.

No receptor, a reconstrução da seqüência $\{x_n\}$ é obtido pela adição de $\{{}^*d_n\}$ ao valor da seqüência anteriormente reconstruída $\{{}^*x_{n-1}\}$:

$$^{*}x_n = {}^{*}x_{n-1} + {}^{*}d_n \tag{6.13}$$

Vamos supor que tanto o transmissor quanto o receptor iniciam suas seqüências pelo mesmo valor $\{x_0\}$, ou seja, $^{*}x_0 = x_0$. Considerando as primeiras interações temos que:

$$d_1 = x_1 - x_0 \tag{6.14}$$

$$^{*}d_1 = Q[d_1] = d_1 + q_1 \tag{6.15}$$

$$^{*}x_1 = x_0 + {}^{*}d_1 = x_0 + d_1 + q_1 = x_1 + q_1 \tag{6.16}$$

$$d_2 = x_2 - x_1 \tag{6.17}$$

$$^{*}d_2 = Q[d_2] = d_2 + q_2 \tag{6.18}$$

$$^{*}x_2 = {}^{*}x_1 + {}^{*}d_2 = x_1 + q_1 + d_2 + q_2 \tag{6.19}$$

$$^{*}x_2 = x_2 + q_1 + q_2 \tag{6.20}$$

Fazendo-se isso para “n” iterações:

$$^{*}x_n = x_n + \sum_{k=1}^{n} q_k \tag{6.21}$$

Nós podemos ver que o erro de quantização se acumula no decorrer do processo. Note que o codificador e o decodificador estão operando com diferentes pedaços de informação. O codificador gera a seqüências de diferenças baseadas nos valores das amostras originais, enquanto o decodificador adiciona a diferenças quantizadas com a versão distorcida do sinal original. Nós podemos resolver este problema forçando ambos codificador e decodificador a usarem a mesma informação durante a operação diferencial e de reconstrução.

A única informação utilizável no receptor que diz respeito a seqüência $\{x_n\}$ é a seqüência reconstruída $\{^{*}x_n\}$. Como essa informação também é utilizada no transmissor, pode-se modificar a operação diferencial para usar o valor reconstruído das amostras anteriores, por que dessa forma tem-se:

$$d_n = x_n - {}^{*}x_{n-1} \tag{6.22}$$

Usando está nova operação diferencial, vamos repetir nosso exame sobre os processos de quantização e reconstrução. Assumiremos que ${}^{*}x_0 = x_0$.

$$d_1 = x_1 - x_0 \tag{6.23}$$

$${}^{*}d_1 = Q[d_1] = d_1 + q_1 \tag{6.24}$$

$${}^{*}x_1 = x_0 + {}^{*}d_1 = x_0 + d_1 + q_1 = x_1 + q_1 \tag{6.25}$$

$$d_2 = x_2 - x_1 \tag{6.26}$$

$${}^{*}d_2 = Q[d_2] = d_2 + q_2 \tag{6.27}$$

$${}^{*}x_2 = {}^{*}x_1 + {}^{*}d_2 = x_1 + q_1 + d_2 + q_2 \tag{6.28}$$

$${}^{*}x_2 = x_2 + q_1 + q_2 \tag{6.29}$$

$${}^{*}x_2 = {}^{*}x_1 + {}^{*}d_2 = {}^{*}x_1 + d_2 + q_2 \tag{6.30}$$

$${}^{*}x_2 = x_2 + q_2 \tag{6.31}$$

Depois de “n” interações teremos:

$${}^{*}x_n = x_n + q_n \tag{6.32}$$

E desta forma não haverá acumulação do ruído de quantização e, portanto será reduzido esse sinal de ruído.

O bloco do sistema de codificação diferencial pode ser visualizado na Figura 6.3.

O sistema de codificação diferencial como o DPCM tem sua eficiência vinculada a redução na variância e extensão dinâmica .A redução da variância depende da capacidade do preditor de predizer o próximo símbolo baseado no passado dos símbolos reconstruídos.

Como um DPCM consiste de dois componentes, o quantizador e o preditor, fazer o DPCM adaptativo significa fazer o quantizador e o preditor adaptativo. Isso quer dizer que iremos adaptar um sistema baseado em sua entrada ou saída.

Esquemas de codificação diferencial são imensamente populares para voz, eles são usados em sistemas de telefone, mensagem de voz e aplicações multimídia.

O codificador DPCM Adaptativo faz parte de vários padrões internacionais (ITU-T G.721,ITU G.723, ITU G.726, ITU-T G.722). Essas recomendações assumem que a voz é amostrada a uma taxa de 8.000 amostras por segundo, assim as taxas de 40,32,24 e 16 kbits por segundo, ou de outra forma, 5, 4, 3 e 2 bits por amostras.

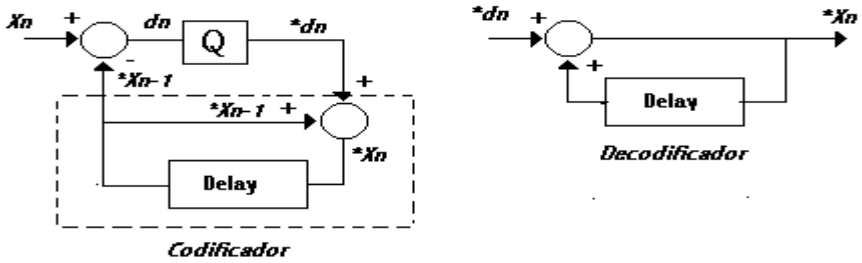

Figura 6.3 – Sistema de Codificação Diferencial

6.4.2 Predição em DPCM

Sistemas de codificação diferencial como DPCM tem como vantagem na variância e a extensão da seqüência diferencial. A redução da variância depende da performance do preditor no quesito predizer os próximos símbolos baseados na reconstrução dos anteriores. Para alcançar o objetivo de projetar um preditor, é necessário o conhecimento dos conceitos matemáticos de expectativa e correlação:

$$\sigma_d^2 = E\left[(x_n - p_n)^2\right] \tag{6.33}$$

σ_d - Variância da Seqüência Diferencial

p_n - Valor da predição

E[ ] – Operador Expectativa

Um bom preditor essencialmente seleciona uma função que minimize σ_d^2. Um dos problemas é que $^*x_n$ é dado por:

$$^*x_n = x_n + q_n \tag{6.34}$$

e q_n é dependente da variância de d_n. Desta forma, ao modificar $f(.)$, nos afetamos σ_d^2, que por sua vez afetará a reconstrução $^*x_n$, que então afetará a seleção de $f(.)$. Para evitar isso, assumiremos a ocorrência de uma quantização fina.

Dessa forma, assume-se que os passos de quantização são tão pequenos que pode-se substituir ${}^{*}x_n$ por x_n, portanto:

$$p_n = f(x_{n-1}, x_{n-2}, ..., x_0) \tag{6.35}$$

Uma vez descoberto a função $f(.)$, podemos usá-la conjuntamente com o valor reconstruído para obter p_n. Se assumirmos que a saída da fonte é um processo estacionário, saberemos que a função que minimizará σ_d^2 é uma expectativa condicional $E(X_n | X_{n-1}, X_{n-2}, ..., X_0)$.

Determinar está expectativa condicional requer o conhecimento de "n" probabilidades condicionais, o que não é adequado.

Dado a dificuldade de encontrar a melhor solução, em muitas aplicações simplifica-se o problema restringindo a função preditora a ser linear. Isto é, a predição p_n passa a ser:

$$p_n = \sum_{i=1}^{N} a_i \cdot {}^{*}x_{n-i} \tag{6.36}$$

O valor de N especifica a ordem do preditor. Usando a quantização fina, pode-se descrever o problema de projetar o preditor como sendo o seguinte: encontrar $\{a_i\}$ de tal maneira que minimize σ_d^2.

$$\sigma_d^2 = E\left[\left(x_n - \sum_{i=1}^{N} a_i \cdot {}^{*}x_{n-i}\right)^2\right] \tag{6.37}$$

Fazendo a derivada de σ_d^2 em relação a cada $\{a_i\}$ e igualando o conjunto a zero teremos N equações e N incógnitas. Para isso considerou-se que o processo é estacionário.

$$\sum_{i=1}^{N} a_i \cdot R_{xx}(i-1) = R_{xx}(1) \tag{6.38}$$

$$\sum_{i=1}^{N} a_i \cdot R_{xx}(i-2) = R_{xx}(2) \tag{6.39}$$

$$\vdots$$

$$\sum_{i=1}^{N} a_i \cdot R_{xx}(i-N) = R_{xx}(N) \tag{6.40}$$

onde $R_{xx}(k)$ é a função de auto-correlação de X_n:

$$R_{xx}(k) = E[x_n x_{n+k}] \tag{6.41}$$

Escrevendo esta equação na forma matricial como :

$$[R]\cdot[A]=[P] \tag{6.42}$$

onde,

$$R=\begin{bmatrix} R_{xx}(0) & R_{xx}(1) & R_{xx}(2) & \cdots & R_{xx}(N-1) \\ R_{xx}(1) & R_{xx}(0) & R_{xx}(1) & \cdots & R_{xx}(N-2) \\ R_{xx}(2) & R_{xx}(1) & R_{xx}(0) & \cdots & R_{xx}(N-3) \\ \vdots & \vdots & \vdots & & \vdots \\ R_{xx}(N-1) & R_{xx}(N-2) & R_{xx}(N-3) & \cdots & R_{xx}(0) \end{bmatrix} \tag{6.43}$$

$$A=\begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_N \end{bmatrix} \tag{6.44}$$

$$P=\begin{bmatrix} R_{xx}(1) \\ R_{xx}(2) \\ R_{xx}(3) \\ \vdots \\ R_{xx}(N) \end{bmatrix} \tag{6.45}$$

Então para encontrar os coeficientes de predição é preciso descobrir os valores da auto-correlação $\{R_{xx}(k)\}$ e em seguida faz-se:

$$[A]=[R^{-1}]\cdot[P] \tag{6.46}$$

6.4.3 DPCM Adaptativo

Pode-se adaptar um sistema a partir de sua entrada ou da saída de dados, sendo que o primeiro se chama *forward adaptation,* o último *backward adaptation.* No caso de *forward adaptation* , os parâmetros do sistema são atualizados baseados na entrada do codificador. Não é possível realizar este processo no decodificador. Portanto, os parâmetros atualizados são enviados para o decodificador juntamente com a informação. Em *forward adaptation* a entrada de dados é dividida em blocos, sendo que os parâmetros do quantizador são calculados por cada bloco.

No caso de *backward adaptation, a adaptação é* baseada na saída do decodificador. Como o processo é adequado para ser usado no decodificado não é necessário fazer a transmissão dos parâmetros.

6.5 Vocoders

Ao contrário dos codificadores de forma de onda, os vocoders utilizam uma modelagem de como o sinal foi gerado e a partir disso tentam extrair do sinal codificado esses parâmetros que em seguida são enviados para o decodificador. Pode-se chegar a conclusão de que vocoders são feitos para determinados sinais gerados, ou seja, seu desempenho cai quando ele processa sinais diferentes dos de voz.

Vocoders assumem um explicito modelo de produção de voz que assume o sinal como sendo produzido por uma excitação de um sistema linear (o trato vocal) através de uma série periódica de pulsos. Se a voz consiste de uma série de impulsos, à distância entre esses pulsos é igual ao período pitch.

O modelo do sistema linear para o trato vocal e seus parâmetros pode ser determinado usando várias técnicas. São justamente esses métodos que determinam a distinção entre os vários tipos de vocoders.

O transmissor de voz é analisado para determinar os parâmetros e a excitação. Esta informação é então transmitida para o receptor onde a voz é sintetizada. O resultado disso é que ele pode produzir sinal de voz inteligível para baixa taxa de bits. A pobre qualidade da saída do vocoder é atribuída a natureza muito simples do modelo de produção de voz.

6.5.1 Vocoder de Canal

No canal do vocoder, cada segmento na entrada de voz é analisada usando um banco de filtros passa faixa. A energia na saída de cada filtro é estimada em intervalos fixos e transmitida para o receptor. Na implementação digital a energia estimada pode ser o valor médio quadrático. Por meio da estimativa da saída do filtro, é feita a decisão sobre a natureza da voz, ou seja, se ela é um sinal sonoro**,** como nos casos dos sons de /a/ /e/ /o/, ou sinal ruidoso, como no caso dos sons/s/ /f/. Em um sinal sonoro tende-se a ter estruturas pseudo-periódicas. O período do harmônico fundamental é chamado de período *pitch*. No transmissor é feita uma estimativa do período *pitch* que é enviado para o receptor.

No receptor, o filtro do trato vocal é implementado por um banco de filtros passa faixa. O banco de filtros conhecido como filtros sintetizadores é idêntico ao banco de filtros analisadores.

O modelamento é feito através da representação do filtro sintetizador como um filtro variante no tempo que é excitado por uma fonte de ruído branco, para sinais surdos (aleatórios) ou um trem de pulso separado por um período *pitch* para sinais sonoros.Portanto os parâmetros a serem transmitidos serão:

- Parâmetros do Filtro
- Sinalização informando o tipo de sinal
- Ganho
- Caso o sinal seja sonoro, o período do trem de pulso (período *pitch*).

Os parâmetros do filtro podem ser determinados pelo codificador tanto no domínio do tempo quanto da freqüência.

6.5.2 LPC – Linear Predictive Coder

No LPC o trato vocal é modelado como um único filtro linear da qual a saída y_n é relatada de uma entrada ε_n por:

$$y_n = \sum_{i=1}^{M} b_i \cdot y_{n-1} + G \cdot \varepsilon_n \tag{6.47}$$

onde G é chamado de ganho do filtro.

Como no caso do vocoder de canal, a entrada para o filtro do trato vocal é também a saída de um gerador de ruído aleatório ou um gerador de pulso periódico.

No transmissor segmentos da voz são analisados. Geralmente, esse processo de segmentações segue o seguinte critério: a entrada é amostrada numa taxa de 8000 amostras por segundo que então é dividida em segmentos. Sendo assim cada segmento de voz terá uma duração de 22.5 milisegundos. A partir desses segmentos são obtidos os parâmetros sobre a natureza da voz (sinal sonoro ou ruidoso). Isso é feito a partir de duas observações: as amostras com grande amplitude são de sinal sonoro (ou seja, há mais energia) e amostras com alta freqüência são sinais ruidosos.

Um outro parâmetro é o período *pitch* que pode ser estimado a partir de diferentes algoritmos. Vários deles fazem uso do fato de que a autocorrelação de uma função periódica $R_{XX}(k)$ possui valor máximo quando *k* é igual ao período *pitch*.

Em seguida é obtido o filtro do trato vocal, o qual é modelado por um filtro linear com uma relação entrada-saída que obedece à Eq. 6.36. Os coeficientes deste filtro são calculados minimizando o erro entre a predição e a amostra atual. Isso é feito de maneira idêntica a feita na predição DPCM. Outro método existente é o método da covariância.

Após determinar os coeficientes, eles são quantizados e transmitidos para os receptores. Esse sinal de voz então passa através do inverso do filtro do trato vocal para obter o erro de predição ou residual.

No receptor, os frames do sinal sonoro são obtidos pela excitação do filtro receptor do trato vocal por uma forma de onda armazenada. Se o frame é do sinal ruidoso**,** o filtro sintetizador é excitado por um gerador de números pseudo–aleatórios. O fato de haver apenas dois tipos de sinais de excitação dá uma qualidade artificial da voz. Esta aproximação não é adequada quando usado em ambiente ruidoso.

7. Procedimentos Experimentais

O projeto teve como objetivo demonstrar e avaliar métodos de compressão e comparar processos com perdas e sem perdas de dado, dessa forma esse capítulo pretende demonstrar o funcionamento e a performance do processo de compressão de áudio. Foram escolhidos dois programas para serem testadas empiricamente, os quais seguiram as mesmas linhas de estudo observadas na parte teórica.

O sistema operacional utilizado para executar os programas foi o Linux Red Hat 7.0 e o programa utilizado para compilar e executar os códigos foi o gcc. No caso da Codificação de Huffman o gcc foi executado através da interface gráfica KDE 2.1 enquanto que no caso do DPCM o gcc foi executado a partir do Console.

Esse procedimento adotado foi de grande êxito, porém entre as dificuldades encontradas destacamos a inexperiência com o ambiente operacional utilizado.

Para se analisar a performance dos arquivos de áudio originais em relação aos reconstruídos pelos decodificadores foram utilizados o programa Sound Forge 5.0, do qual pode-se capturar as imagens do sinal no domínio do tempo e da freqüência. Entretanto esse seria utilizado para uma análise mais visual, onde seria muito difícil ter uma conclusão sobre a qualidade da reconstrução.

Além desse teste, com o programa distau.exe pode-se comparar dois arquivos de áudio em relação as medidas de distorção. E por último foi feito um teste subjetivo sobre a qualidade da reconstrução em relação a original.

Foram escolhidos dois algoritmos para serem testados empiricamente, os quais seguiram a mesma linha de estudo observada na parte teórica.

Para todos esses testes foram utilizados dois tipos de arquivos: um arquivo tipo texto e um arquivo de áudio (trecho da música My Girl - The Temptations). Comprimiu-se esses arquivos das seguintes maneiras:

- Música codificada pelo algoritmo DPCM com 8 níveis de quantização e distribuição Gaussiana.
- Música codificada pelo algoritmo DPCM com 8 níveis de quantização e distribuição Laplaciana.
- Música codificada pelo algoritmo de Huffman.
- Texto codificado pelo algoritmo de Huffman.

Para se utilizar a compressão com o algoritmo DPCM fez-se dois "Makefiles", responsáveis pela compilação do código fonte a partir do compilador gcc, e criação do executável chamado dpcm_enc. Em seguida, a partir do Console foi executada a seguinte linha de comando como se pode visualizar a seguir:

```
[root@hermes root]# cd Desktop/PERFEITOS/dpcm_enc
[root@hermes dpcm_enc]# dpcm_enc -i MUSICA.RAW -o MUSICA.DPCM -q ga08.dat -p coeficientes.dat
-s 1
Number of levels: 8
Number of data points is 1036288
File MUSICA.RAW opened for read
Number of data points is 1036288
num 1036288 ave 0.300589 var 5040376.000000 std 2245.078125
mean and variance of residuals 0.000754   0.241770
Signal to noise ratio 13.793232   Signal to prediction error ratio 6.166493
```

Sendo assim, o executável dpcm_enc comprimiu o arquivo de entrada (MUSICA.RAW) e gerou o arquivo comprimido (MUSICA.DPCM). Como se pode observar, o parâmetro (-q ga08.dat) foi utilizado para abrir o arquivo de entrada ga08.dat. Este arquivo possui os parâmetros do quantizador, nesse caso, 8 níveis de quantização com distribuição Gaussiana. O fator de compressão foi de 81%, já que o arquivo MUSICA.RAW de 2 Mbytes resultou no MUSICA.DPCM de 379,5 kbytes.

Dando prosseguimento, utilizou-se novamente o Console para fazer a descompressão a partir do executável dpcm_dec**.** Da mesma forma, foi necessário compilar o programa dpcm_dec.c e gerar o executável para em seguida executar a descompressão a partir da linha de comando abaixo:

```
[root@hermes root]# cd Desktop/PERFEITOS/dpcm_dec

[root@hermes dpcm_dec]# dpcm_dec -i MUSICA.DPCM -o MUSICA_ga08_RECONSTRUIDA.RAW

Quantizer file is ga08.dat

 Number of levels: 8

Predictor file is coeficientes.dat

Number of data points is 1036288

bits/label = 3

Average value = 0.300589, Standard deviation = 2245.078125, scale = 1.000000
```

Isso resultou num arquivo decodificado (MUSICA_ga08_RECONSTRUIDA.RAW) com 2 Mbytes. Com os dois arquivos em mãos analisou-se a performance a partir da comparação dos sinais original e reconstruído, representados aqui na Figura7.1 e na Figura7.2, respectivamente.

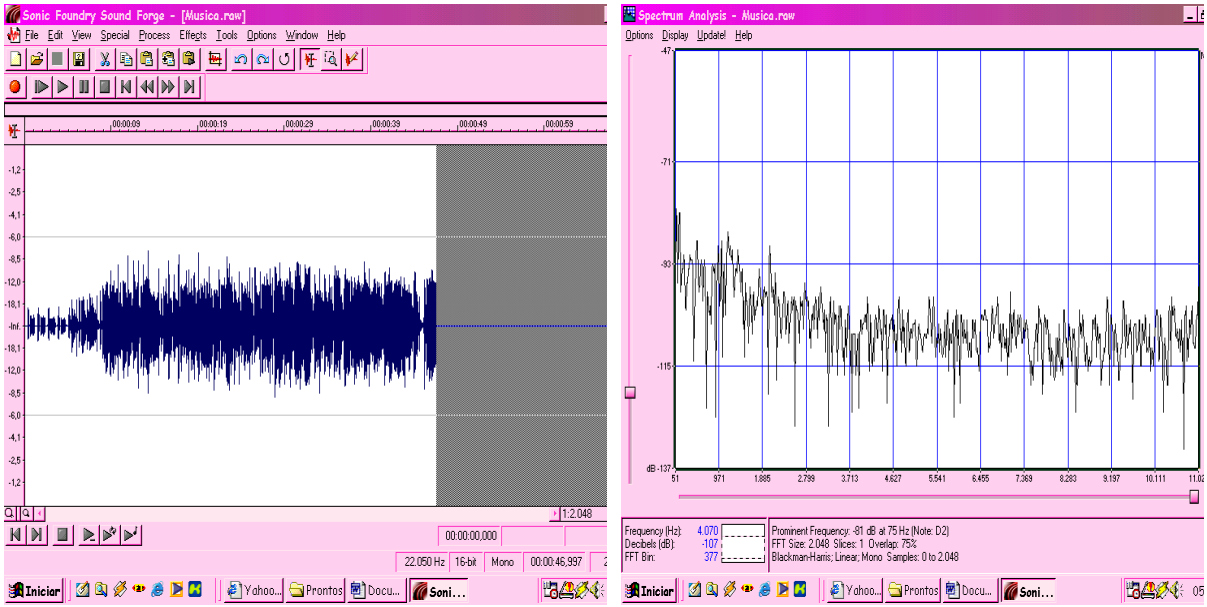

Figura 7.1 – Sinal do arquivo MUSICA.RAW (original) no tempo e na freqüência

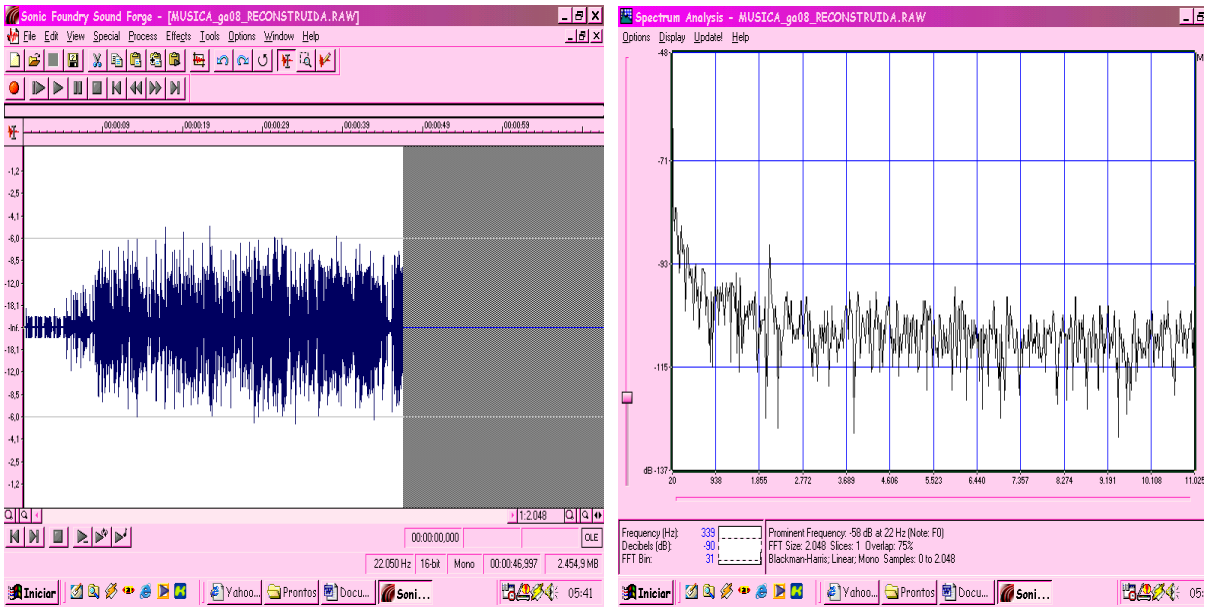

Figura 7.2 – Sinal do arquivo reconstruído (Distribuição Gaussiana) no tempo e na freqüência.

Para se fazer a análise quantitativa executou-se o programa distau que comparou os dois arquivos disponibilizando os seguintes dados:

```
[root@hermes distau]# distau MUSICA.RAW MUSICA_ga08_RECONSTRUIDA.RAW

Number of data points is 1036288

File MUSICA.RAW opened for read

Number of data points is 1036288

File MUSICA_ga08_RECONSTRUIDA.RAW opened for read

Number of data points is 1036288

mean squared error = 327416.44,

mean absolute error = 421.35

Signal to noise ratio 11.873625
```

O mesmo teste foi realizado para a compressão DPCM utilizando a distribuição Laplaciana. A Figura 7.3 apresenta o sinal reconstruído no domínio do tempo e da freqüência. Em seguida, procedeu-se a análise quantitativa com a ajuda do programa distau.exe.

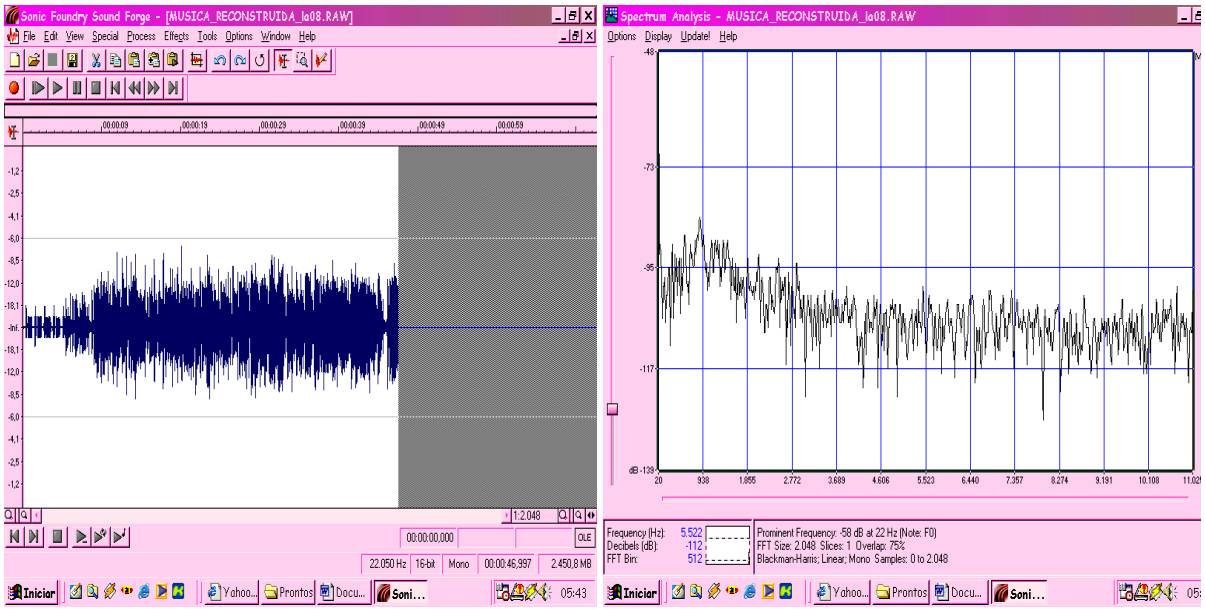

Figura 7.3 – Sinal do arquivo reconstruído (Distribuição Laplaciana) no tempo e na freqüência.

Os dados gerados pela comparação entre as amostras entre o arquivo original e o reconstruído foram os seguintes:

```
[root@hermes distau]# distau MUSICA.RAW MUSICA_RECONSTRUIDA_la08.RAW

Number of data points is 1036288
```

```
File MUSICA.RAW opened for read

Number of data points is 1036288

File MUSICA_RECONSTRUIDA_la08.RAW opened for read

Number of data points is 1036288

mean squared error = 131267.67

mean absolute error = 312.02

Signal to noise ratio 15.843052
```

Mudando o método de compressão, será analisada agora a performance do algoritmo de Codificação de Huffman, começando pela visualização do sinal reconstruído (Figura7.4).

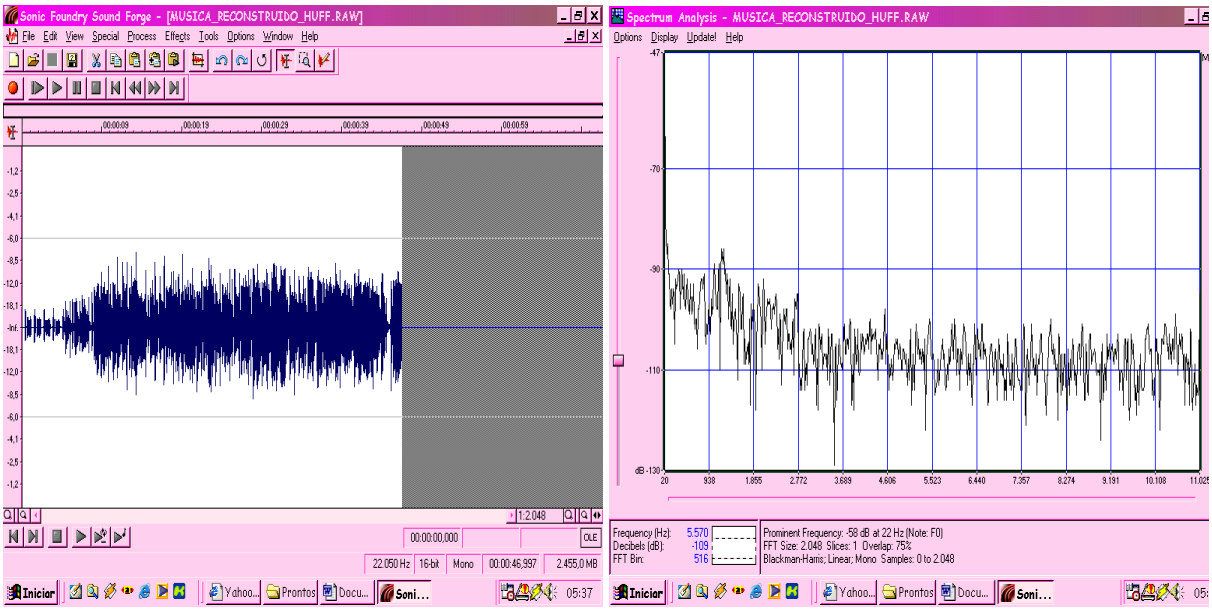

Figura 7.4 – Sinal do Arquivo reconstruído (Codificação de Huffman) no tempo e na freqüência.

Os dados gerados pelo executável distau.exe a partir da comparação entre as amostras do arquivo original e do reconstruído foram os seguintes:

```
[root@hermes distau]# distau MUSICA.RAW MUSICA_RECONSTRUIDO_HUFF.RAW

Number of data points is 1036288

File MUSICA.RAW opened for read

Number of data points is 1036288

File MUSICA_RECONSTRUIDO_HUFF.RAW opened for read

Number of data points is 1036287

mean squared error =  0.06,
```

```
mean absolute error =  0.00

Signal to noise ratio 79.014633
```

Por último será analisado um arquivo de texto. O algoritmo DPCM não processa esse tipo de arquivo, por isso o teste será realizado apenas para a codificação de Huffman.

Através do teste foi possível constatar que o texto original permaneceu intacto após a compressão e descompressão, o que era esperado já que o codificador de Huffman é um tipo de compressão sem perdas. Salientando-se que o arquivo original possui 10,5 kbytes e o reconstruído 5,8 kbytes, ou seja, houve uma compressão de 44,8%.

Os dados gerados pelo executável distau.exe a partir da comparação entre as amostras do arquivo original e do reconstruído foram os seguintes:

```
[root@hermes distau]# distau TEXTO.WOR TEXTO_RECONSTRUIDO_HUFF.WOR

Number of data points is 5372

File TEXTO.WOR opened for read

Number of data points is 5372

File TEXTO_RECONSTRUIDO_HUFF.WOR opened for read

Number of data points is 5372

mean squared error =  0.00

mean absolute error =  0.00

Signal to noise ratio inf
```

Com esses dados em mãos é possível fazer uma tabela comparativa:

Tabela 7.1: Comparação da Performance

Tipo de Arquivo	Compressão	Distribuição	Erro Quadrático Médio	Avaliação Subjetiva	Taxa de Compressão	SNR
Musica	DPCM	Gaussiano	327416,44	Aceitável	81%	11,87
		Laplaciano	131267,67	Aceitável	81%	15,84
Musica	Huffman	-	0,06	Ótimo	10%	79,01
Texto	Huffman	-	0,00	Ótimo	45%	-

O Arquivo coeficientes.dat contém a ordem do preditor seguido pelos valores dos coeficientes.Já os arquivos ga08.dat e la08.dat carregam os dados referentes a distribuição (gaxx.dat ou laxx.dat), ao número de níveis de quantização (xx08.dat), aos limites de cada nível e aos valores de reconstrução.

8. Conclusões

Esse trabalho aborda os aspectos que rodeiam o processo de compressão de áudio. Com esse intuito, dividiu-se o trabalho em duas etapas, uma teórica e que trouxe todo o embasamento e outro prático, que apenas confirmou nossas expectativas.

A parte teórica se propôs a expor dois temas, a compressão onde há perdas de dados e a compressão sem perdas. Pretendeu-se fazer uma apresentação de fácil entendimento e que demonstra-se exemplos de aplicações dos métodos, os quais mais tarde passaria a ser avaliados empiricamente. Portanto o texto tentou proporcionar ferramentas matemáticas e estatísticas, de retratar metodologias e de em seguida estudar casos onde se pode implementá-las.

Também houve subdivisões na exposição dessas técnicas, como é o caso do DPCM onde existiram variações no que se refere à distribuição escolhida. O tipo de distribuição é uma forma de se tentar predizer as amostras, sendo assim quando se diz que foi utilizada a distribuição laplaciana, na verdade quer indicar que a fonte de informação possui uma distribuição de suas amostra que mais se assemelha a uma distribuição laplaciana.

No caso do arquivo de música se verificou um melhor desempenho da distribuição Laplaciana como pode ser comprovado pelos dados obtidos através do executável distau.exe. Neles se evidencia tanto um menor erro médio quadrático (mean square error) como um menor erro médio absoluto, isso conseqüentemente leva a esperar uma maior SNR para a distribuição laplaciana.

Na codificação de Huffman fez uma subdivisão que levou em conta o tipo de arquivo processado, ou seja, fez-se uso de um arquivo de texto e um de áudio. Isso foi feito pois já se esperava um melhor rendimento da codificação de Huffman para o caso de um texto. Pode-se explicar isso da seguinte maneira, na codificação de Huffman é transmitida a árvore binária que posiciona os símbolos em suas folha através do critério de ocorrência. Portanto é de se esperar que numa árvore que possua número de símbolos igual à quantidade de letras e números (isto é, igual ao número de caracteres ASCII) seja menor do que uma que possua símbolos que representem os níveis de reconstrução, é claro que se pode usar poucos níveis de reconstrução como foi o caso aqui. Contudo também foi utilizado um texto curto, onde conseqüentemente alguns caracteres não estavam presentes.O DPCM não processa arquivos de texto e por isso não está apto a se avaliado.

Os resultados demonstram a grande diferença que consiste em usar cada uma das duas técnicas. O algoritmo de Huffman processou o sinal e o reconstruiu integralmente, porem o fator de compressão deixou a desejar. Todavia para aplicações onde a fidelidade é a principal preocupação, ele seria

fortemente recomendável, visto que os dados obtidos sobre sua performance quantitativa retratam um erro quadrático e absoluto médio praticamente nulo.

Na avaliação subjetiva no qual se ouviu o sinal reconstruído e o comparou com o original, a codificação de Huffman obteve um resultado perfeito, tanto para o texto quanto para Música.

O DPCM demonstrou bom desempenho para atividades computacionais onde a confiabilidade nos dados transmitidos não seja ponto principal. Obteve-se um fator de compressão de 81% para ambas distribuições. Ficou evidente ao comparar os fatores de potencia, no caso DPCM foi de 81% e no Huffman 10%.

Pode-se também se fazer uma avaliação teórica sobre o LPC (Linear Prediction Coding). Ele é uma boa solução quando se possui o objetivo primário de se obter uma comunicação inteligível em baixas taxas. É usada uma taxa de 2.4 kbps que foi especificada pelo U.S. Government Standard LPC-10.

9. Referências

[1] Sayood, Khalid, “Introduction a data compression”, Morgan Kaufmann, 2ª edição, ISBN: 1-55860-558-4, 2002.

[2] Tanenbaum, Andrew, “Redes de computadores”, 3ª edição, São Paulo, Ed. Campus, 1998.

[3] Enciclopédia Barsa 2000.

[4] Bordignon, Mário R., “Vídeo Conferência: Conceitos, Tecnologias e Uso”, Book Express, ISBN: 85-86846-77-5, 2001.

[5] Gibson, J. D., Berger, T., Lookabaugh, T., Lindbergh, D. e Baker, R. L., “Digital Compression for Multimedia: Principles and Standards”, Morgan Kaufmann, ISBN: 1-55860-369-7, 1998.

[6] Rao, K. R. e Hwang, J. J., “Techniques and Standards for Image Video and Audio Coding”, Prentice Hall, ISBN: 0-13-309907-5, 1996.

[7] Watkinson, John, MPEG-2, Focal Press, 1ª edição, ISBN: 0 240 51510 2.

[8] Schildt, Herbert, Turbo C Guia do Usuário, McGraw-Hill, 2ª edição.

10. Apêndices

CÓDIGO FONTE DO PROGRAMA DPCM.ENC

```
#include <stdio.h>
#include <math.h>
#include <unistd.h>
#include "idc.h"

void usage(void);

main(int argc, char *argv[])
{
  float mean, var, secmom, ave, std;
  float residual, sig, qerr, msqe, snr, sper;
  float reconstruction, output,resrec,prob[256], entropy;
  int numlev,numbits,label, ent_flag, end_flag, name_length;
  char infile[100], outfile[100], qfile[100], pfile[100];
  float *bound, *recon, *pred, *xf, prediction, scale;
  int i, j, k, num, order, flag, size;
  int c;
  extern int  optind;
  extern char *optarg;
  FILE *ofp,*qfp, *pfp;
  short *aufile;

  order = -1;
  numlev = -1;
  ent_flag = 0;
  ofp = stdout;
  scale = 1.0;
  strcpy(outfile,"standard out");
  qfp = NULL;
  pfp = NULL;

  while((c=getopt(argc,argv,"i:o:q:p:s:eh"))!=EOF)
  {
   switch (c){
   case 'i':
         strcpy(infile,optarg);
         break;
   case 'o':
         strcpy(outfile,optarg);
         if((ofp=fopen(outfile,"wb")) == NULL)
         {
          fprintf(stderr,"Unable to open file %s for write\n",outfile);
          exit(1);
         }
         break;
   case 'q':
         strcpy(qfile,optarg);
         if((qfp=fopen(qfile,"r")) == NULL)
         {
          fprintf(stderr,"Unable to open file %s for reading quantizer
parameters\n",qfile);
          exit(2);
         }
```

```
          break;
   case 'p':
          strcpy(pfile,optarg);
          if((pfp=fopen(pfile,"r")) == NULL)
          {
           fprintf(stderr,"Unable to open file %s for reading predictor
parameters\n",qfile);
           exit(3);
          }
          break;
   case 's':
          sscanf(optarg,"%f",&scale);
          break;
   case 'e':
          ent_flag++;
          break;
   case 'h':
          usage();
          exit(4);
          break;
               }
   }
   if(qfp == NULL)
   {
     fprintf(stderr,"You need to enter the filename containing the
quantizer\n");
     fprintf(stderr,"parameters for this program to work.  Please do so
now.\n");
     scanf("%s",&qfile);
     if((qfp=fopen(qfile,"r")) == NULL)
      {
       fprintf(stderr,"Unable to open file %s for reading quantizer
parameters\n",qfile);
       exit(4);
      }
    }

/*                write quantizer file name to output file              */


   name_length = strlen(qfile);
   fwrite(&name_length,1,sizeof(int),ofp);
   fwrite(qfile,name_length+1,sizeof(char),ofp);



  fscanf(qfp,"%d",&numlev);
  fprintf(stderr," Number of levels: %d\n",numlev);

  bound = (float *) calloc((numlev+1),sizeof(float));
  recon = (float *) calloc((numlev+1),sizeof(float));



  for(i=0;i<numlev-1;i++)
   fscanf(qfp,"%g",&bound[i]);

  for(i=0;i<numlev;i++)
   fscanf(qfp,"%g",&recon[i]);
```

```
    if(pfp == NULL)
    {
     fprintf(stderr,"\nWe are assuming a single tap predictor with
prediction\n");
     fprintf(stderr,"coefficient 0.86.  If you want to use a different
predictor\n");
     fprintf(stderr,"run this program with the -p option\n\n");
     order = 1;
     pred = (float *) calloc(1,sizeof(float));
     pred[0] = 0.86;
     strcpy(pfile,"nofile");
    }
    else
    {
     fscanf(pfp,"%d",&order);
     pred = (float *) calloc(order,sizeof(float));
     for(i=0; i<order; i++)
       fscanf(pfp,"%g",&pred[i]);
    }


/*               write predictor file name to output file             */

   name_length = strlen(pfile);

   fwrite(&name_length,1,sizeof(int),ofp);
   fwrite(pfile,name_length+1,sizeof(char),ofp);


/* get filesize for the audio file */

   get_file_size(infile,&size);

/* assuming it takes two bytes for each audio or speech sample */

  num = size/2;
  fprintf(stderr,"Number of data points is %d\n",num);
  fwrite(&num,1,sizeof(int),ofp);

/* get memory for the audio or speech file   */

  aufile = (short *) calloc(num,sizeof(short));

/*   Get the audio data */

  readau(infile,aufile);


/* Convert the data into float */

  xf = (float *) calloc(num,sizeof(float));
  for(i=0; i< num; i++)
    xf[i] = (float) aufile[i];

/* Normalize  */

  norm(xf,num,&ave,&std);

  fwrite(&ave,1,sizeof(float),ofp);
  fwrite(&std,1,sizeof(float),ofp);
  fwrite(&scale,1,sizeof(float),ofp);
```

```
   numbits = (int) (log((double) numlev)/log((double) 2.) + 0.99999);


  mean = 0.0;
  secmom = 0.0;
  sig = 0;
  msqe = 0.0;
   if(ent_flag)
    for(i=0;i<numlev;i++)
      prob[i] = 0.0;


 ave = 0;
 secmom = 0;
 end_flag = 0;
 for(i=0; i< num; i++)
 {
  if(i==(num-1))
   end_flag = 1;
  predictor(i,order,pred,xf,&prediction);
  sig += xf[i]*xf[i];
  residual = xf[i] - prediction;
  ave += residual;
  secmom += residual * residual;
  residual = scale*residual;
  label = nuquan_enc(residual, bound, numlev);
  if(ent_flag)
      prob[label]++;
  stuffit(label,numbits,ofp,end_flag);
  resrec = nuquan_dec(label,  recon);
  resrec = resrec/scale;
  reconstruction = prediction + resrec;
  msqe += (xf[i]-reconstruction)*(xf[i]-reconstruction);
  xf[i] = reconstruction;
 }
 ave = ave /(float) (num - order);
 secmom = secmom / (float) (num - order);
 var = secmom - ave*ave;
 printf("mean and variance of residuals %f   %f\n",ave,var);
 sig = sig/ (float) (num - order);
 msqe = msqe/ (float) (num - order);
 snr = 10.0 * log10(sig/msqe);
 sper = 10.0 * log10(sig/secmom);

 printf("Signal to noise ratio %f   Signal to prediction error ratio
%f\n",snr,sper);

if(ent_flag)
   {
    for(i=0;i<numlev;i++)
      prob[i] = prob[i]/(float) num;
    entropy = ent(prob,numlev);
    printf("The output entropy is %f\n",entropy);
   }

}

void usage(void)
{
```

```
  fprintf(stderr,"Usage:\n dpcm_enc [-i infile][-o outfile][-q qfile][-p
pfile][-s scale][-e][-h]\n");
  fprintf(stderr,"\t infile : file containing the audio or speech
input\n");
  fprintf(stderr,"\t outfile: file containing the compressed output\n");
  fprintf(stderr,"\t qfile  : file containing the quantizer parameters\n");
  fprintf(stderr,"\t pfile  : file containing the predictor coefficients.
If\n");
  fprintf(stderr,"\t\t this option is not used the program uses a one tap
predictor\n");
 fprintf(stderr,"\t\t with predictor coefficient 0.86\n");
 fprintf(stderr,"\t scale   : scales the input to the quantizer.\n");
 fprintf(stderr,"\t -e  : This option results in the computation of the
output entropy\n");
 fprintf(stderr,"\t\t of the dpcm encoder.\n");
 fprintf(stderr,"\t -h  : results in the display of this message\n");
}
```

CÓDIGO FONTE DO PROGRAMA DPCM.DEC

```
#include <stdio.h>
#include <math.h>
#include <unistd.h>
#include <malloc.h>
#include "idc.h"

void usage(void);

main(int argc, char *argv[])
{
  float ave, std, reconstruction, output,resrec, scale;
  int numlev,numbits,label, name_length;
  char infile[100], outfile[100], qfile[100], pfile[100];
  float *bound, *recon, *pred, *xf, prediction;
  int i, j, k, num, order, count, size, *buffer;
  int c;
  extern int  optind;
  extern char *optarg;
  FILE *ifp, *ofp,*qfp, *pfp;
  short *aufile;

  order = -1;
  numlev = -1;
  ofp = stdout;
  strcpy(outfile,"standard out");
  qfp = NULL;
  pfp = NULL;
  while((c=getopt(argc,argv,"i:o:h"))!=EOF)
  {
   switch (c){
   case 'i':
         strcpy(infile,optarg);
         if((ifp=fopen(infile,"rb")) == NULL)
         {
          fprintf(stderr,"Unable to open file %s for read\n",infile);
          exit(1);
         }
         break;
   case 'o':
         strcpy(outfile,optarg);
         if((ofp=fopen(outfile,"wb")) == NULL)
```

```
          {
           fprintf(stderr,"Unable to open file %s for write\n",outfile);
           exit(1);
          }
          break;
   case 'h':
          usage();
          exit(4);
          break;
               }
   }

/*                read quantizer file name */


   fread(&name_length,1,sizeof(int),ifp);
   fread(qfile,name_length+1,sizeof(char),ifp);
   fprintf(stderr,"Quantizer file is %s\n",qfile);


     if((qfp=fopen(qfile,"r")) == NULL)
      {
       fprintf(stderr,"Unable to open file %s for reading quantizer
parameters\n",qfile);
       exit(4);
      }



  fscanf(qfp,"%d",&numlev);
  fprintf(stderr," Number of levels: %d\n",numlev);

  bound = (float *) calloc((numlev+1),sizeof(float));
  recon = (float *) calloc((numlev+1),sizeof(float));



  for(i=0;i<numlev-1;i++)
   fscanf(qfp,"%g",&bound[i]);

  for(i=0;i<numlev;i++)
   fscanf(qfp,"%g",&recon[i]);




/*                read predictor file name              */


   fread(&name_length,1,sizeof(int),ifp);
   fread(pfile,name_length+1,sizeof(char),ifp);

   fprintf(stderr,"Predictor file is %s\n",pfile);

   if(!strcmp(pfile,"nofile"))
    {
     order = 1;
     pred[0] = 0.866;
     fprintf(stderr,"No predictor file specified.  A first order
predictor\n");
     fprintf(stderr,"with predictor coefficient 0.86 will be used\n");
```

```
    }
   else
     if((pfp=fopen(pfile,"r")) == NULL)
     {
      fprintf(stderr,"Unable to open file %s for reading predictor
parameters\n",pfile);
      exit(3);
     }
     else
     {
     fscanf(pfp,"%d",&order);
     pred = (float *) calloc(order,sizeof(float));
     for(i=0; i<order; i++)
       fscanf(pfp,"%g",&pred[i]);
     }


/* get filesize for the audio file */

   fread(&num,1,sizeof(int),ifp);

/* assuming it takes two bytes for each audio or speech sample */

  fprintf(stderr,"Number of data points is %d\n",num);
  fread(&ave,1,sizeof(float),ifp);
  fread(&std,1,sizeof(float),ifp);
  fread(&scale,1,sizeof(float),ifp);

    buffer = (int *) calloc(num+8,sizeof(int));

/* Read the coded speech or audio file */

    numbits = (int) (log((double) numlev)/log((double) 2.) + 0.99999);
    fprintf(stderr,"bits/label = %d\n",numbits);
    unstuff(numbits,ifp,buffer,&count);
    if(count != num)
    {
     fprintf(stderr,"Mismatch in amount of data\n");
     fprintf(stderr,"num = %d, Number of data points read =
%d\n",num,count);
    }

/* get memory for the audio or speech file   */

  aufile = (short *) calloc(num,sizeof(short));

  xf = (float *) calloc(num,sizeof(float));



  fprintf(stderr,"Average value = %f, Standard deviation = %f, scale =
%f\n",ave,std,scale);


 for(i=0; i< num; i++)
 {
  predictor(i,order,pred,xf,&prediction);
  label = buffer[i];
  resrec = nuquan_dec(label,recon);
  resrec = resrec/scale;
  reconstruction = prediction + resrec;
  xf[i] = reconstruction;
```

```
 }

/*   Denormalize  */

   for(i=0; i<num; i++)
     xf[i] = xf[i]*std+ave;

/* write out to file  */


   for(i=0; i<=num; i++)
   {
     if(xf[i] > 32767)
       xf[i] = 32767;
     if(xf[i] < -32768)
       xf[i] = -32768;
     aufile[i] = (short) xf[i];
   }

   if(fwrite(aufile,sizeof(short),num,ofp)!=num)
     fprintf(stderr,"what the ....");
}

void usage(void)
{

  fprintf(stderr,"Usage:\n dpcm_dec [-i infile][-o outfile][-e][-h]\n");
  fprintf(stderr,"\t infile : file containing the compressed speech or
audio\n");
  fprintf(stderr,"\t outfile: file containing the reconstructed output\n");
 fprintf(stderr,"\t -h  : results in the display of this message\n");
}
```

CÓDIGO FONTE DO PROGRAMA HUFF.ENC

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <string.h>
/* define the structure node */
typedef struct {
                float pro; /* probabilities */
                int l; /* location of probability before sorted */
                unsigned int code; /* code */
                struct node *left; /* pointer for binary tree */
                struct node *right; /* pointer for binary tree */
                struct node *forward;
                struct node *back;
                struct node *parent; /* pointer to parent */
                int check;
} node;
typedef struct node NODE;
typedef struct node *BTREE;
#include "idc.h"
#include "unistd.h"
#include "imsub.c"
#include "sub.c"
void usage(void);

void main(int argc, char **argv)
{
```

```
     unsigned char *file; /*pointer to an array for file */
     char infile[80], outfile[80], codefile[80], scodefile[80];
               /* input and output files*/
     char temp [80],type,where,t;
     int size,num,c;
     int i,j,n; /* counters */
     FILE *ifp, *ofp, *cfp, *sfp, *tmp_fp;
     char *length,x; /*pointer to an array for code lengths*/
     int values[256], loc[256];
     unsigned int *code;
          /* pointer to an array for code */
     float prob[256],p;
     extern int optint;
     extern char *optarg;

     ifp=stdin;
     t=0;  /*flag to see if an input filename was given*/
     ofp=stdout;
     x=0; /* flag if output is piped to decoder */
     cfp=NULL;
     sfp=NULL;
     num=256;

        code=(unsigned int *)malloc(num*sizeof(unsigned int));
        length=(char *)malloc(num*sizeof(char));

     while((c=getopt(argc,argv,"i:o:c:s:h"))!=EOF)
     {
     switch(c){

/* input file */

     case 'i':
          strcpy(infile,optarg);
          if((ifp=fopen(infile,"rb"))==NULL){
                 fprintf(stderr,"Image file cannot be opened for input.\n");
                 return;
          }
          t=1;
          break;

/* output file */

     case 'o':
          strcpy(outfile,optarg);
          if((ofp=fopen(outfile,"wb"))==NULL){
                 fprintf(stderr,"Output file cannot be opened for
output.\n");
                 return;
          }
          x=1;
          break;
/* code file */

     case 'c':
          strcpy(codefile,optarg);
                 if((cfp=fopen(codefile,"rb"))==NULL){
                 fprintf(stderr,"Code file cannot be opened for input.\n");
                 return;
          }
          getcode(cfp,num,code,length);
                 break;
```

```
/* file to store code in */

     case 's':

          strcpy(scodefile,optarg);
                if((sfp=fopen(scodefile,"wb"))==NULL){
                fprintf(stderr,"Code file cannot be opened for output.\n");
                return;
          }
          break;

        case 'h':
              usage();
              exit(1);
              break;
          }
     }

/* get size of file */

   /* create a temporary file for input */

     if(t==0){
          strcpy(infile,"tmpf");
                tmp_fp=fopen(infile,"wb+");
                while((t=getc(ifp))!=EOF)
                        putc(t,tmp_fp);
                fclose(tmp_fp);
                ifp=fopen(infile,"rb");
          t=0;
     }

     fseek(ifp,0,2); /* set file pointer at end of file */
     size=ftell(ifp); /* gets size of file */
     ++size;
     fseek(ifp,0,0); /* set file pointer to begining of file */

/* get memory for file */

         file=(unsigned char*)malloc(size*sizeof(unsigned char));
         if (file==NULL){
                printf("Unable to allocate memory for file.\n");
                exit(1);
     }

/* get file */

         fread(file,sizeof(unsigned char),size,ifp);
         fclose(ifp);

/* remove temporary file if one was used */

         if(t==0)
                remove("tmpf");

/* create code */

     if(cfp==NULL){

/* set values to zero */
```

```
        for(i=0;i<num;i++)
                values[i]=0;

/* find values */

     value(values,file,size,num);

/* find probs */

     p=size+0.0;
     for(i=0;i<num;i++)
          prob[i]=values[i]/p;

/* set to zero */

     for(i=0;i<num;i++){
          code[i]=0;
          length[i]=0;
     }

/* sort prob array */

     sort(prob,loc,num);

/* make huff code */

     huff(prob,loc,num, code, length);

     }

/* encode file */

     size=files(size,code,length,file);

/* write length of encoded file to the decoder */

/*   if(x==0)
     fwrite(&size,sizeof(int),1,ofp);
*/

     if(sfp==NULL){

/* write encoded file to file */

             fwrite(code,sizeof(unsigned int),num,ofp);
          fwrite(length,sizeof(char),num,ofp);
     }

        fwrite(file,sizeof(unsigned char),size,ofp);
        fclose(ofp);

/* write code to a file */

     if(sfp!=NULL){
          fwrite(code,sizeof(unsigned int),num,sfp);
          fwrite(length,sizeof(char),num,sfp);
          fclose(sfp);
     }

}

void usage()
```

```
{
  fprintf(stderr,"Usage:\n");
  fprintf(stderr,"huff_enc [-i infile][-o outfile][-c codefile][-s
storecode][-h]\n");
  fprintf(stderr,"\t imagein : file containing the input to be encoded.  If
no\n");
  fprintf(stderr,"\t\t name is provided input is read from standard in.
This\n");
   fprintf(stderr,"\t\t feature can be used to directly encode the output
of programs\n");
   fprintf(stderr,"\t\t such as jpegll_enc, and aqfimg_enc.\n");
  fprintf(stderr,"\t outfile : File to contain the encoded representation.
If no\n");
  fprintf(stderr,"\t\t name is provided the output is written to standard
out.\n");
  fprintf(stderr,"\t codefile: If this option is used the program will read
the\n");
   fprintf(stderr,"\t\t Huffman code from codefile.  If the option is not
used the\n");
  fprintf(stderr,"\t\t program computes the Huffman code for the file being
encoded\n");
  fprintf(stderr,"\t storecod: If this option is specified the Huffman code
used to\n");
  fprintf(stderr,"\t\t encode the file is stored in codefile.  If this
option is\n");
   fprintf(stderr,"\t\t not specified the code is stored in outfile\n");
}
```

CÓDIGO FONTE DO PROGRAMA HUFF.DEC

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <unistd.h>
#include "idc.h"

void usage(void);

void main(int argc, char **argv)
{
     unsigned char *file,*efile;
          /*pointer to an array for file
            pointer to an array for encoded image*/
     unsigned char w,p,n; /* place holders */
     char infile[80], outfile[80], codefile[80]; /* input and output
files*/
     char temp [80],where,size;
     int num,s,x,c;
     int i,k,l,count,f; /* counters */
     FILE *ifp,*ofp,*cfp,*tmp_fp;
     unsigned int *code,word,d;
```

```
     char *length,t;  /* pointer to an array for code lengths */
     extern int optint;
     extern char *optarg;

     ifp=stdin;
     t=0; /* flag to see if an input filename was given */
     ofp=stdout;
     cfp=NULL;
     num=256;
     l=0;
     word=0;
        n=1<<7; /* set highest bit equal to one */
     size=0;
     f=10000;
     x=0;

/* get memory */

        code=(unsigned int *)malloc(num*sizeof(unsigned int));
        length=(char *)malloc(num*sizeof(char));
        file=(unsigned char *)malloc(f*sizeof(unsigned char));
        if (file==NULL){
                printf("Unable to allocate memory for image.\n");
                exit(1);
     }

        while((c=getopt(argc,argv,"i:o:c:h"))!=EOF)
        {
        switch(c){

/* input file */

        case 'i':
                strcpy(infile,optarg);
                if((ifp=fopen(infile,"rb"))==NULL){
                printf("Cannot open file for input!\n");
                return;
          }
          t=1;
          break;
```

```
/* output file */

        case 'o':
                strcpy(outfile,optarg);
                ofp=fopen(outfile,"wb");
                break;

/* code file */

        case 'c':
                strcpy(codefile,optarg);
                cfp=fopen(optarg,"rb");
                getcode(cfp,num,code,length);
                break;
        case 'h':
                usage();
                exit(1);
                break;
     }
     }

   fprintf(stderr,"\t Patience -- This may take a while\n");

/* get file size */

   /* create a temporary file for input */

     if(t==0){
          strcpy(infile,"tmpf");
          tmp_fp=fopen(infile,"wb+");
          while((t=getc(ifp))!=EOF)
               putc(t,tmp_fp);
          fclose(tmp_fp);
                ifp=fopen(infile,"rb");
          t=0;
     }

     fseek(ifp,0,2); /* set file pointer at end of file */
     s=ftell(ifp); /* gets size of file */
     fseek(ifp,0,0); /* set file pointer to begining of file */
```

```
     if(cfp==NULL){
          fread(code,sizeof(unsigned int),num,ifp);
          fread(length,sizeof(char),num,ifp);
          s=s-ftell(ifp); /* s = the size of the encoded file */
          }

/* get memory for encoded file */

     efile=(unsigned char *)malloc(s*sizeof(unsigned char));

/* get encoded file */

     fread(efile,sizeof(unsigned char),s,ifp);
     fclose(ifp);

/* remove temporary file if one was used */

     if(t==0)
          remove("tmpf");

/* decode file */

     count=0;
     w=*(efile+count); /* w equals encoded word */
     ++count; /* counter to keep track of which word is being decoded */
    for( ;count<s; ){
        for(k=0;k<8;k++){
          if((w & n) == n){ /* checks bit value of encoded word */
               ++word;
               }
          ++size; /* counter to keep track of length of word */

     /* check to see if word equals a code */

          for(i=0;i<num;i++){
               if(word==code[i] && size==length[i]){
                    if(l<f){
                         p=i; /* pixal value is a char */
                         *(file+l)=p; /* decoded image */
                         ++l; /* counter of length of image */
                         i=num; /* ends loop */
```

```
                    }
                else{
                    fwrite(file,sizeof(unsigned char),f,ofp);
                    l=0;
                                    p=i; /* pixal value is a char */
                                    *(file+l)=p; /* decoded image */
                                    ++l; /* counter of length of image
*/
                                    i=num; /* ends loop */
                    x=1;
                }
                word=0; /* reset word */
                size=0; /* reset size */
                        }
            }
    word<<=1;
    w<<=1;
         }
         w=*(efile+count); /* get next encoded value */
         ++count;
    }
    *(file+l)=EOF;
    fwrite(file,sizeof(unsigned char),l,ofp);
    fclose(ofp);

}

void usage()
{
  fprintf(stderr,"Usage:\n");
  fprintf(stderr,"huff_enc [-i infile][-o outfile][-c codefile][-h]\n");
  fprintf(stderr,"\t imagein : file containing the Huffman-encoded data.
If no\n");
  fprintf(stderr,"\t\t name is provided input is read from standard in.\n"
);
  fprintf(stderr,"\t outfile : File to contain the reconstructed
representation.  If no\n");
  fprintf(stderr,"\t\t name is provided the output is written to standard
out.\n
");
```

```
  fprintf(stderr,"\t codefile: This option is required if the Huffman
encoded file\n");
  fprintf(stderr,"\t\t does not contain the Huffman code as part of the
header\n");
  fprintf(stderr,"\t\t If this option is specified the program will read
the\
n");
   fprintf(stderr,"\t\t Huffman code from codefile.\n");
}
```

CÓDIGO FONTE DO PROGRAMA DISTAU

```
#include <stdio.h>
#include <math.h>
#include <unistd.h>
#include "idc.h"

void usage(void);

main(int argc, char *argv[])
{
   int size, num, i;
   float error, sqrerror, abserror, sigpower, mse, mae, sig, snr;
   short *aufile1, *aufile2;

/* get filesize for the first audio file */

   get_file_size(argv[1],&size);

  num = size/2;
  fprintf(stderr,"Number of data points is %d\n",num);

/* get memory for the audio or speech file   */

  aufile1 = (short *) calloc(num+8,sizeof(short));
  aufile2 = (short *) calloc(num+8,sizeof(short));

/*   Get the audio data */

  readau(argv[1],aufile1);
  readau(argv[2],aufile2);


/* Compute performance metrics */

  sqrerror = 0.0;
  abserror = 0.0;
  sigpower = 0.0;

  for(i=0; i<num; i++)
  {
   sig   = (float) aufile1[i];
   error = (float) (aufile1[i]-aufile2[i]);
   sqrerror += error*error;
   abserror += myabs(error);
   sigpower += sig*sig;
  }
```

```
  mse = sqrerror/ (float) num;
  mae = abserror/ (float) num;
  printf("mean squared error = %5.2f, mean absolute error =
%5.2f\n",mse,mae);
  sig = sigpower/ (float) num;
  snr = 10.0 * (float) (log((double) (sig/mse))/log((double) 10.));

 printf("Signal to noise ratio %f\n",snr);
}

void usage(void)
{

  fprintf(stderr,"Usage:\n distau file1 file2\n");
  fprintf(stderr,"\t file1 : file containing the audio or speech input\n");
  fprintf(stderr,"\t file2 : file containing the reconstructed output\n");
  fprintf(stderr,"\t -h  : results in the display of this message\n");
}
```