INSTITUTO SUPERIOR DE
ENGENHARIA DE LISBOA

ADEETC

Licenciatura em Engenharia Informática e Multimédia

Semestre de Verão

2º Trabalho Prático

Codificação de Sinais Multimédia

Realizado por:
João Santos nº 39348
Rui Santos nº 39286

Abril, 2017

Conteúdo

Lista de Figuras

Lista de Tabelas

1 Introdução

O segundo trabalho prático da disciplina de CSM (Codificação de Sinais Multimédia) tem como objetivo a exploração dos conceitos de compressão de dados sem perdas baseados na teoria da informação.
Foi utilizado o algoritmo de *Huffman* no trabalho prático para obter uma codificação sem perdas. Foram implementadas várias funções que permitiram realizar o trabalho e que vão ser descritas mais à frente. Para testar a robustez do algoritmo foram utilizados vários tipos de média, tais como, imagens, texto (txt e pdf), audio (mp3), midi e um ficheiro com extensão .txt que contem um ecg (eletrocardiograma). Com os algoritmos necessários desenvolvidos e testados foi realizado o cálculo da entropia e a interpretação dos resultados obtidos.
Finalmente houve uma comparação entre os ficheiros codificados e descodificados, sendo que, o objetivo é que ambos os ficheiros sejam idênticos pois a compressão implementada é *lossless* (sem perdas).

2 Desenvolvimento

A fase de desenvolvimento do trabalho conta com a compreensão do algoritmo de *Huffman* e um estudo sobre o modo mais eficiente de desenvolver os métodos necessários. Para que o compressor seja bem desenvolvido foi pedido no enunciado do trabalho a implementação dos seguintes métodos:

1. função **gera_huffman()** que gera uma tabela com o código binário para cada símbolo de um dado conjunto, usando o método de *Huffman*. Esta função tem como parâmetros de entrada um conjunto de símbolos e o número de ocorrências de cada símbolo, dado pelo seu histograma.

2. função **codifica**() que dada uma mensagem (sequência de símbolos) e a tabela da ponto anterior, retorne uma sequência de bits com a mensagem codificada.

3. função **descodifica()** que dada uma sequência de bits (mensagem codificada) e a tabela do ponto 1, retorne uma sequência de símbolos (mensagem descodificada).

4. função **escrever()** que dada uma sequência de bits (mensagem codificada) e o nome do ficheiro, escreva a sequência de bits para o ficheiro.

5. função **ler()** que dado o nome do ficheiro, leia uma sequência de bits (mensagem codificada) contida no ficheiro.

Podemos observar na figura 1 um diagrama com o fluxo de ocorrências do programa desenvolvido.

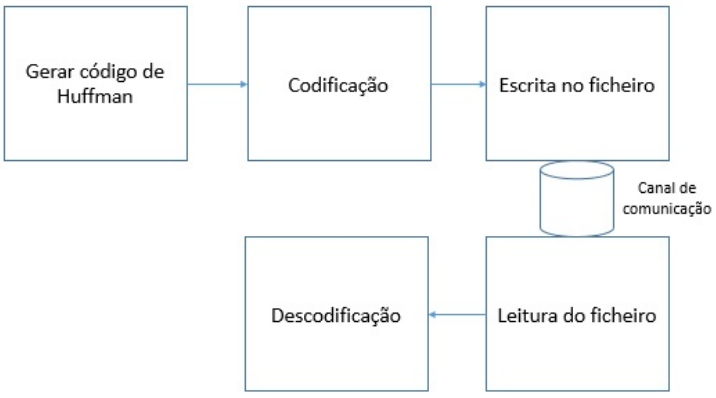

Figura 1: Diagrama do compressor de Huffman.

2.1 Algoritmo de Huffman

A codificação de *Huffman* é um método de compressão que utiliza as probabilidades de ocorrência dos símbolos no conjunto de dados a comprimir, de modo, a determinar códigos de comprimento variável (VLC) para cada símbolo. Este algoritmo é utilizado na compressão JPEG e MPEG.

2.1.1 Algoritmo de codificação de Huffman

O modo de funcionamento do algoritmo é o seguinte:

1. **Ordenar** os símbolos de acordo com a respetiva frequência de ocorrência ou probabilidade.

2. Aplicar, de forma recursiva, um processo de **contração** aos dois símbolos com menor probabilidade, obtendo-se um novo símbolo cuja probabilidade é igual à soma das probabilidades dos símbolos que lhe deram origem, repetindo-se esta contração até que todos os símbolos tenham sido contraídos em **um** único símbolo com probabilidade igual a 1.

Para uma melhor compreensão do algoritmo de *Huffman* podemos utilizar a tabela exemplo 1 com os seguintes símbolos e ocorrências:

Símbolo	Ocorrência
A	24
B	12
C	10
D	8
E	8

Tabela 1: Tabela exemplo.

Partindo da tabela 1 é possível representar em forma de árvore os símbolos e as suas ocorrências, obtendo a figura 2.

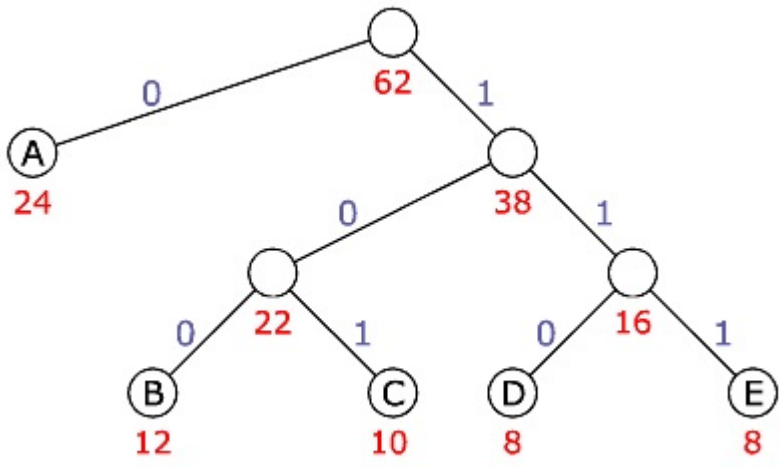

Figura 2: Árvore de código de acordo com Huffman.

Como é possível observar através do diagrama em árvore os símbolos com maior número de ocorrências vão ser codificados com menor número de bits, para este exemplo o caso mais percetível é o símbolo *A* que é codificado com apenas um bit. Com a árvore desenvolvida e os bits atribuídos a codificação final de cada símbolo estão representados na tabela 2.

Símbolo	Ocorrência	Código	Tamanho do código
A	24	0	1
B	12	100	3
C	10	101	3
D	8	110	3
E	8	111	3

Tabela 2: Tabela final da codificação de Huffman.

Supondo agora, por exemplo, que a mensagem a codificar, é a sequência "ABADE", esta mensagem codificada com a técnica de *Huffman* resulta na seguinte sequência de 11 bits:

01000110111

2.1.2 Descodificação de Huffman

Para ilustrar o algoritmo de **descodificação de bits em série** retome-se o exemplo da figura 1, assumindo que a tabela de codificação binária já se encontra disponível no lado do descodificador.
A descodificação de *Huffman* baseado no algoritmo de descodificação de bits em série consiste nos seguintes passos:

1. Ler o fluxo de bits comprimidos, **bit após bit**, e percorrer a tabela até que se encontre uma ocorrência.

2. À medida que cada bit é utilizado, deverá ser **descartado**. Quando existe uma ocorrência, o descodificador de *Huffman* coloca na saída o **símbolo** correspondente à entrada da tabela, completando a descodificação do símbolo atual.

3. Repetir os passos 1 e 2 até que o fluxo comprimido tenha sido consumido.

2.2 Escrita e Leitura em ficheiro

De modo a simular um canal de comunicação foram desenvolvidos os métodos *escrever()* e *ler()*, como os nomes indicam permitem a escrita e leitura num ficheiro de extensão .txt.

2.2.1 Escrita

O método escrever é um método que passados como argumentos uma sequência de bits (mensagem codificada) e o nome do ficheiro, escreve a sequência de bits para esse ficheiro.
A solução implementada neste método tem quatro pontos fundamentais, dos quais:

1. Realizar um *reshape* na mensagem recebida de modo a agrupar os bits de 8 em 8.

2. Com os bits agrupados de 8 em 8 realizamos uma transformação de valores binários para inteiros.

3. Com a lista de inteiros é realizada uma transformação para caracteres.

4. Escrita dos caracteres no ficheiro.

2.2.2 Leitura

O método de leitura *ler()* faz exatamente o oposto ao método de escrita, ou seja, dado o nome do ficheiro, é lida a sequência de caracteres (mensagem codificada) contida no ficheiro e transformada para uma mensagem binária. A solução encontrada para este método tem os seguintes pontos fundamentais:

1. Leitura dos caracteres do ficheiro.
2. Conversão de caracteres para números inteiros.
3. Conversão de inteiros para binário.

2.3 Resultados obtidos

Com todas as funções desenvolvidas é nos pedido no enunciado do trabalho a realização de cálculos e a medição de tempos de processamento de modo a testar o algoritmo. Os pontos pedidos são os seguintes:

a) Gerar o código usando a função **gera_huffman()**. Medição do tempo que demora a função.

b) Medição da entropia e o número médio de bits por símbolo. Calculo da eficiência.

c) Codificação da mensagem contida no ficheiro (usando a função **codifica()**). Medição do tempo que a função demora a fazer a codificação.

d) Gravação de um ficheiro com a mensagem codificada, usando a função **escrever()**. Calculo do tamanho do ficheiro.

e) Ler do ficheiro o conjunto de bits, usando a função **ler()**.

f) Descodificação da mensagem (usando a função **descodifica()**) Medição do tempo que a função demora a fazer a descodificação.

g) Comparação da mensagem descodificada com a original e verificar que são iguais (erro nulo).

2.3.1 Tempos de processamento

Os tempos de processamento são obtidos utilizando a biblioteca *time* e são calculados do seguinte modo:

```
tempo1 = time()
funcao()
tempo2 = time()
tempo_processamento = tempo2 - tempo1
```

Podemos observar na tabela 3 os tempos de processamento obtidos para cada função desenvolvida e o tempo total do algoritmo.

Função	Tempo de processamento (*segundos*)
gera_huffman()	0.003
codifica()	0.220
escrever()	2.299
ler()	0.301
descodifica()	12.730
Tempo total	15.559

Tabela 3: Tabela com os tempos de processamento.

Como se pode observar a partir da tabela 3 os tempos de processamento são relativamente baixo, exceto, o do descodificador. Isto deve-se ao facto de ser necessário realizar uma comparação bit a bit da mensagem e verificar se esse valor binário existe na tabela. O tempo total de processamento ronda os 15 segundos primeiramente devido ao tempo do descodificador.

2.3.2 Entropia

A entropia pode ser pode ser definida como o número médio mínimo de *bits* necessário à representação de cada símbolo (S_i) de uma dada fonte (S) em função da probabilidade de ocorrência desse símbolo na fonte. Logo, a entropia é tanto maior quanto mais a incerteza. Podemos observar a fórmula utilizada para calculo da entropia em baixo.

$$H(S) = -\sum_{i=1}^{N} p(s_i) \log_2 p(s_i)$$

Como o código de Huffman respeita, a entropia da fonte, os símbolos com maior probabilidade serão codificados com menos bits e os símbolos menos recorrentes serão codificados com mais *bits*, pelo que o código final resultante é, em média, muito menor do que a fonte.

2.3.3 Número médio de bits

De seguida foi realizado o calculo do número médio de *bits* por símbolo, como o nome indica, é uma média do número de *bits* necessários para codificar cada símbolo. Podemos observar a formula utilizada para o calculo do número médio de *bits* em baixo.

$$L = \sum_{i=1}^{N} p(s_i)L(s_i)$$

2.3.4 Eficiência

Por último com a entropia e o número médio de bits por símbolo calculados é possível calcular a eficiência do código através da seguinte fórmula.

$$\eta = \frac{H(S)}{L}$$

O teorema de codificação de fonte de *Shannon* diz que é possível codificar uma fonte, sem perdas, com um número médio de bits por símbolo maior ou igual à entropia da fonte:

$$H(S) < L < H(S) + \delta$$

Assim, a codificação entrópica, consiste em codificar uma fonte de forma a atingir um número médio de bits por símbolo igual à entropia dessa fonte.

2.3.5 Valores obtidos

Entropia	7.445
Nº médio de bits	7.467
Eficiência	0.996

Tabela 4: Tabela com os valores obtidos.

Como referido anteriormente o valor da entropia e do número médio de *bits* por símbolo deve ser igual à entropia. Observando os valores obtidos podemos deduzir que isto se confirma. Como a eficiência do código é a divisão entre estes dois valores e estes são quase idênticos é esperado uma eficiência de código com valor aproximado de 1 que vai de acordo com o resultado obtido.

2.3.6 Ficheiros obtidos

Por fim resta representar a comparação dos ficheiros antes e depois da descodificação. Visto ser uma técnica de compressão sem perdas os tamanhos dos ficheiros devem ser idênticos e a sua taxa de compressão deve dar uma valor igual a 1. A taxa de compressão é dada pela expressão:

$$\text{Taxa de Compressão} = \frac{\text{Imagem Original}}{\text{Imagem Descodificada}}$$

Utilizando a biblioteca **path** e o método *getsize()* dessa mesma biblioteca conseguimos obter os tamanhos das imagens originais e descodificadas, respetivamente. Através dos resultados obtidos podemos calcular a Taxa de Compressão utilizando a expressão descrita em cima. É possível observar na tabela 5 os resultados obtidos.

Tamanho da imagem original	210122
Tamanho da imagem descodificada	210122
Taxa de Compressão	1

Tabela 5: Tamanhos das imagens e Taxa de Compressão.

Como seria de esperar visto que a codificação de *Huffman* se trata de uma codificação sem perdas o tamanho da imagem original e da imagem codificada tinha que ser idêntico, tal como, se pode observar na tabela 5. Sendo ambos os tamanhos idênticos a Taxa de Compressão irá ter o valor desejado de 1.

Por último, podemos observar a imagem resultante da descodificação em comparação com a imagem original, observemos então as figuras 3 e 4.

Figura 3: Imagem original.

Figura 4: Imagem descodificada.

3 Conclusões

Com a finalização do trabalho foram obtidos conhecimentos acerca de técnicas com compressão sem perdas (*Lossless*). Estas técnicas permitem fazer uma recuperação total dos dados após a compressão e devido a isso é obtida uma baixa taxa de compressão. Foram também estudados conceitos como a quantidade de bits necessária para representar um símbolo, entropia de fonte, número médio de bits por símbolo e eficiência do código. Os resultados obtidos estão dentro do que era suposto observar tirando o facto de apenas ter sido utilizada um tipo de média para teste do compressor desenvolvido.