Universidade de Aveiro

**Informação e Codificação Projeto 2**



Rafael Amorim (98197), Diogo Fontes (98403), Renato Ourives (98576)

Departamento de Electrónica, Telecomunicações e Informática

30 de Outubro de 2022

# Conteúdo

1. [Introdução](#_bookmark0) 1
   1. [Compilação](#_bookmark1) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1
2. [Bitstream](#_bookmark2) 3
3. [Codec de Áudio](#_bookmark4) 6
   1. [Preditor](#_bookmark5) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6
   2. [Lossless Codec](#_bookmark6) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7
   3. [Lossy Codec](#_bookmark9) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8
   4. [Resultados](#_bookmark11) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9
4. [Contribuição dos autores](#_bookmark30) 18

**Lista de Figuras**

* 1. [Estrutura lógica do Codificador](#_bookmark7) . . . . . . . . . . . . . . . . . . . . . . . . . . 8
  2. [Estrutura lógica do *Decoder*](#_bookmark8). . . . . . . . . . . . . . . . . . . . . . . . . . . . 8

**Capítulo 1**

# Introdução

O presente relatório visa descrever a resolução do Projecto 2 desenvolvido no âmbito da unidade curricular de Informação e Codificação. O principal objetivo deste projeto era criar dois codificadores, um de imagem e outro de áudio com a possibilidade de a codificação ser efectuada de forma *lossless* ou *lossy*.

## Compilação

O código desenvolvido para o projeto encontra-se disponível em: [https://github.com/Torrakanor611/IC\_project2.](https://github.com/Torrakanor611/IC_project2)

*#Estando no directório base do projeto*

mkdir build && cd build

*#Compilar o código*

cmake .. make

*#Gerar e compilar documentação doxygen*

doxygen doxyconfig

Os diretório do projeto incluem:

* diretório *src* -> como quatro pastas em que cada uma contém classes para implementar módulos do projeto (Bitstream, Golomb, codec de áudio e codec de imagem).
* diretório *test* -> com ficheiros cpp que implementam testes aos módulos desenvolvidos (testes aos codecs, teste ao golomb, bitstream, etc).
* diretório *files* -> com recursos que podem ser usados nos testes (imagens, ficheiros de áudio, etc).

Todos os ficheiros da pasta test, têm comentado no seu topo o comando que permite a sua compilação com as respetivas flags corretas.

**Capítulo 2**

# Bitstream

Esta classe permite manipular um ficheiro dado bit a bit utilizando uma *stream*. O tipo primitivo mais pequeno possível representar C é o *char* e ocupa 1 byte em memória. Assim sendo, a menor unidade suportado em I/O é também 1 *byte*. No entanto, para acções de manipulação de ficheiros, neste caso compressão e descompressão, foi preciso criar esta classe de abstracção para I/O num ficheiro para manipular os ficheiros *bitwise*.

A assinatura pública desta classe é:

BitStream();

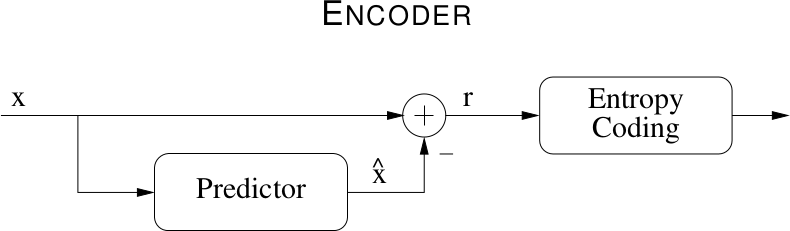
BitStream(const char\* filename, char mode); unsigned char readBit();

void writeBit(char bit);

void readNbits(char\* bits, int n); void writeNbits(char\* bits, int n); bool eof();

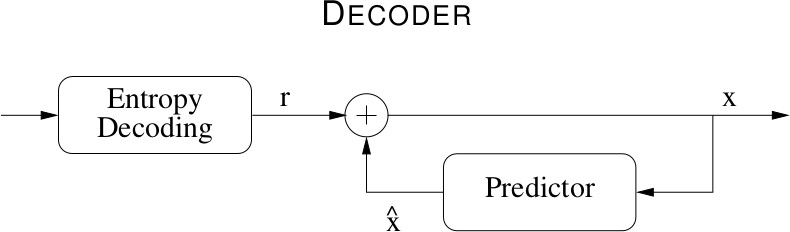
void close();

Um exemplo de uso desta classe é o ficheiro *copybinfile.cpp* na diretório *examples*.



**Figure 4.1:** Estrutura lógica do Codificador

Por sua vez na figura [4.2](#_bookmark8) podemos observar o processo contrário ao que foi descrito an- teriormente que é o de descodificação. Este processo é iniciado ao ler primeiro os cabeçalhos com as funções para tal efeito desenvolvidas na classe *Golomb*. Ao ler o cabeçalho obtemos não só o *m* mas também outros dados que nos permitem reconstruir o ficheiro original e saber quantos mais valores temos de ler. Tendo processado o número de samples, e o valor do *m*, precisamos agora de ler e descodificar todos os valores dos residuais calculados. Essa leitura é feita com sucessivas chamadas à função *decode* da classe *Golomb*.



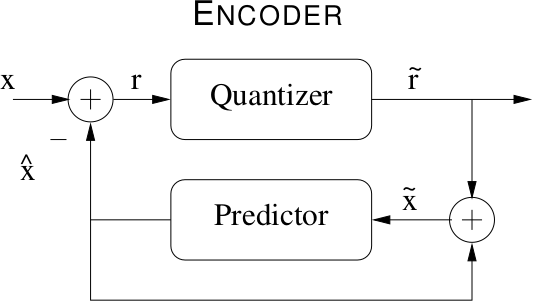
**Figure 4.2:** Estrutura lógica do *Decoder*

Após todo este processo estar concluido são copiados todos os dados obtidos no proces- samento de informação para um ficheiro àudio de output com recurso aos métodos da classe *libsndfile*.

No caso do codec lossy, a estrutura base é a mesma na medida em que são usados os mesmos preditores para codificação e descodificação de dados, assim como os mesmos métodos para obter o melhor valor de *m* para a codificação *Golomb*. Apenas foi feita uma alteração, que foi a adição de um processo de quantização aos residuais calculados com o preditor. No entanto foi também necessário alterar o preditor deste codec pelo facto de o cálculo dos residuais ter de refletir esta quantização que é realizada. Assim sendo sempre que era quantizado um residual, o valor usado nessa posição para o preditor é dado pelo soma de uma predição da sample com o seu residual quantizado.

A quantização é realizada com um shift à direita do número de bits indicado pelo utilizador. Quando se atualiza o valor da sample para fazer o sincronismo do erro (soma da predição com residual quantizado), é desfeito o shift (realizado na direcção oposta o mesmo número de vezes).

O processo de descodificação é precisamente igual ao que é realizado no caso do codec lossless com a exceção de que é necessário reconstruir a quantização que foi efectuada. Essa reconstrução é feita efectuando um shift na direção oposta o mesmo número de vezes que foi feito na codificação.



**Figure 4.3:** Estrutura lógica do Codificador do Codec Lossy

## Resultados

Ao longo da próxima secção (e nos resultados apresentados no próximo capítulo) são apresentadas diversas vezes a taxa de compressão que foi obtida através da seguinte fórmula:

*UncompressedFile*

*Compression Rate* =

*CompressedFile*

Na [Tabela 4.1](#_bookmark12) é possível observar os resultados de vários testes efectuados para testar o codec de áudio lossless. Para além dos resultados dos testes é apresentado também o teste

de compressão para o codec de áudio FLAC para que pudéssemos comparar os resultados obtidos com alguma referência.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Ficheiro** | **Preditor** | **Taxa de Compressão** | **Duração (s)** | **FLAC Compressão** | **FLAC Duração** |
|  | 1 | 1,358 | 1.2070 |  |  |
| sample01.wav | 2  3 | 1,409  1,367 | 1.1427  1.1979 | 1,499 | 0,193 |
|  | 1 | 1,385 | 0.5896 |  |  |
| sample02.wav | 2  3 | 1,377  1,317 | 0.5945  0.5947 | 1,588 | 0,157 |
|  | 1 | 1,466 | 0.7587 |  |  |
| sample03.wav | 2  3 | 1,526  1,500 | 0.7029  0.6975 | 1,780 | 0.183 |

Tabela 4.1: Resultados do Codec Lossless

Em relação aos resultados das taxas de compressão foram bastante parecidos com os do FLAC, mas mesmo assim ficaram sempre abaixo do valor do codec. Comparando os resultados dos preditores, podemos concluir que aquele que teve a melhor taxa de compressão foi nos três casos o preditor 2.

Por sua vez a duração do processo de compressão já foi bastante superior ao tempo do FLAC em todos os casos. Em termos de comparação entre preditores os resultados são sempre bastante semelhantes não se destacando um dos outros por resultados melhores. Podemos também observar que o tempo que demorou a comprimir a sample01.wav foi maior que nos dois casos. Isto deve-se pelo facto de este ser maior que os outros ficheiros e consequentemente mais tempo demora a compressão. Tal facto também se refletiu nos tempos de compressãoo do FLAC.

Analisando agora a entropia antes e depois do processo de predição do codec lossless (no caso do preditor 2 pois este foi o que apresentou melhores resultados), podemos concluir que o preditor cumpriu com o seu objetivo. A principal finalidade de usar um preditor era obter uma sequência de valores cuja a entropia fosse menor que a sequência de valores original. [[3]](#_bookmark33) A entropia neste caso representava o número de símbolos binários necessários para codificar a fonte de informação. Logo sendo que o preditor reduziu a entropia, significa que são necessários menos bits para codificar os símbolos do ficheiro de áudio o que por sua implica que podemos atingir uma maior compressão.

|  |  |  |
| --- | --- | --- |
| **Ficheiro** | **Entropia Original** | **Entropia Residuais** |
| sample01.wav | 13.9086 | 11.3135 |
| sample02.wav | 13.035 | 11.2006 |
| sample03.wav | 13.2678 | 10.3173 |

Tabela 4.2: Ficheiros de texto e a sua entropia

Podemos também observar que a sample03.wav foi a que registou uma entropia de residuais mais baixa. Esse facto é comprovado pela [Tabela 4.1](#_bookmark12) em que se verifica que a sample03.wav foi dos 3 aquele que obteve uma maior taxa de compressão.

Comparando os resultados obtidos anteriormente com os do codec lossy (escolhendo o preditor 2 e indicando que o número de bits a remover é 2) podemos observar que a taxa de compressão foi maior que a do FLAC e do codec lossless em todos os ficheiros. A duração também foi ligeiramente menor que a do codec lossless assim como a entropia dos residuais. O facto de a entropia ser menor implica consequentemente uma maior taxa de compressão o que é comprovado pelos resultados obtidos.

|  |  |  |  |
| --- | --- | --- | --- |
| **Ficheiro** | **Taxa de Compressão** | **Duração (s)** | **Entropia Residuais** |
| sample01.wav | 1,7102 | 1.0039 | 9.31587 |
| sample02.wav | 1,6632 | 0.5671 | 9.21589 |
| sample03.wav | 1,8858 | 0.6843 | 8.33997 |

Tabela 4.3: Resultados do Codec Lossy escolhendo preditor 2 e removendo 2 bits

Na [Tabela 4.4](#_bookmark13) está representado o SNR obtido ao comparar um ficheiro restaurado (após a compressão do codec lossy) com o seu original. Evidentemente podemos concluir que à medida que removemos mais bits mais pequeno é o SNR. Isto acontece pois ao remover bits estamos a introduzir erros na fonte de informação, o que compromete a sua qualidade e por sua vez aumenta o ruído e consequentemente diminui o SNR. Para os dois ultimos valores do número de bits removidos, o ficheiro descomprimido apresenta quase só ruído.

|  |  |
| --- | --- |
| **Número de Bits**  **removidos** | **SNR** |
| 1 | 74.8448 |
| 2 | 66.3979 |
| 5 | 46.7108 |
| 6 | 40.5911 |
| 11 | 10.3909 |
| 12 | 4.35142 |
| 14 | -8.22758 |
| 15 | -14.813 |

Tabela 4.4: Resultados do Codec Lossy para vários valores de remoção de bits

**Capítulo 6**

# Contribuição dos autores

Todos participaram de forma igual na divisão, discussão e elaboração deste trabalho pelo que a percentagem de contribuição de cada aluno fica:

* Rafael Amorim - 33.33%
* Diogo Fontes - 33.33%
* Renato Ourives - 33.33%

# Bibliografia

1. Armando J. Pinho. *Some Notes For the Course Information and Coding*, pages 62–63. Universidade de Aveiro, 2021.
2. David Solomon. *Data Compression The Complete Reference, 3rd Ed*, pages 59–61. Sprin- ger, 2004.
3. Khalid Sayood. *Introduction to Data Compression, 3rd Ed*, pages 16–17. Elsevier, 2006.
4. https://web.archive.org/web/2016[0127065924/http://www.](http://www.equasys.de/)equasys.de/ colorconversion.html.