



universidade
de aveiro

Universidade de Aveiro

Departamento de Eletrónica, Telecomunicações e Informática

Informação e Codificação

2024/2025

Lab work nº 2

Joaquim Andrade (93432), Leandro Rito (92975), Tiago Mendes
(108990)

Introdução

O código para este trabalho encontra-se no seguinte repositório:
<https://github.com/Strikeneerman/IC-projeto2>

Parte 1 - BitStream

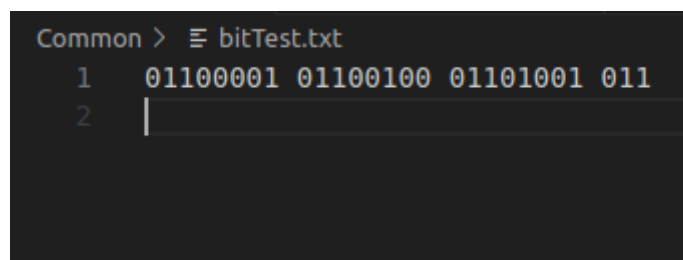
BitStream é uma classe utilizada para manipular fluxos de bits em ficheiros binários. Esta permite a leitura e escrita de bits individuais, assim como sequências de bits e texto.

Para esta classe foi necessário criar as seguintes funções:

- **writeBit**: escreve 1 bit, utilizando também um buffer de 1 byte para especificar que a escrita seja feita byte a byte;
- **readBit**: lê 1 bit, utilizando também um buffer de 1 byte para especificar que a leitura seja feita byte a byte;
- **writeBits**: escreve um número N de bits, entre 1 a 64 bits, utilizando a função writeBit;
- **readBits**: lê um número N de bits, entre 1 a 64 bits, utilizando a função readBit;
- **writeString**: escreve uma string em formato binário, utilizando a função writeBits;
- **readString**: lê uma string, utilizando a função readBits.

Para testar a classe BitStream, foi desenvolvido um programa de teste que permite codificar e decodificar ficheiros de texto:

- A codificação é realizada lendo um ficheiro de texto composto por 1s e 0s (imagem 1). Cada caracter 1 ou 0 é interpretado como um bit, resultando num ficheiro que armazena os valores binários agrupados em bytes, sendo cada byte completo gravado no ficheiro com o valor correspondente em ASCII (imagem 2). Se o número total de bits não for múltiplo de 8, é adicionado bits de padding;
- A decodificação é feita lendo um ficheiro codificado e interpretando os bytes armazenados como sequências de bits, que depois são escritos num ficheiro de texto representando novamente os 1s e 0s originais (imagem 3.)



```
Common > ≡ bitTest.txt
1  01100001 01100100 01101001 011
2  |
```

Imagem 1: ficheiro de teste com valores ASCII em formato de bit

```
Common > ≡ bitEncoded.txt
1  adi |
```

Imagem 2: ficheiro encoded

```
Common > ≡ bitDecoded.txt
1  01100001
2  01100100
3  01101001
4  01100000
5  |
```

Imagem 3: ficheiro decoded

Parte 2 - Golomb

Golomb é uma classe que implementa a codificação e decodificação de Golomb, um método de compressão utilizado para representar números inteiros de forma eficiente que se baseia na decomposição de um número em quociente e resto, dependendo do modo escolhido, sendo estes:

- Modo Interleaved: representa valores inteiros positivos e negativos de forma compacta, utilizando a técnica de codificação zigzag;
- Modo Sign-Magnitude: utiliza um bit adicional para armazenar o sinal do número após a codificação.

Para esta classe foi necessário criar as seguintes funções:

- **zigzagEncode**: converte valores negativos e positivos em valores não negativos;
- **zigzagDecode**: restaura o valor original após a codificação;
- **Encode**: codifica um número inteiro e escreve o resultado num ficheiro binário usando a class BitStream. Este valor é dividido em quociente e resto, com o quociente representado por uma sequência de q bits 1s, seguidos de um 0 e o resto representado de forma binária.

Dependendo do modo selecionado, é utilizada a função zigzagEncode ou é utilizado um bit extra no final para representar o sinal.

- **Decode**: decodifica um número inteiro a partir de um ficheiro binário, lendo o quociente através da interpretação da sequência de bits 1 seguidos de um 0, lendo o resto utilizando o mesmo número de bits usados na codificação e aplicando a fórmula $q*m+r$ para reconstruir o número original.

Dependendo do modo selecionado, é utilizada a função zigzagDecode para restaurar o valor original ou é interpretado um bit extra para o sinal.

Para testar a classe BitStream, foi desenvolvido um programa de teste que mostra os bits gerados pela codificação de Golomb, dependendo se o modo utilizado é Sign-Magnitude (imagem 4) ou Interleaved (imagem 5). Neste caso, o m utilizado foi 5.

```

-5: 10 000 1
-4: 0 100 1
-3: 0 011 1
-2: 0 010 1
-1: 0 001 1
0: 0 000 0
1: 0 001 0
2: 0 010 0
3: 0 011 0
4: 0 100 0
5: 10 000 0
6: 10 001 0
7: 10 010 0
8: 10 011 0
9: 10 100 0
10: 110 000 0
11: 110 001 0
12: 110 010 0
13: 110 011 0
14: 110 100 0
15: 1110 000 0
16: 1110 001 0
17: 1110 010 0
18: 1110 011 0
19: 1110 100 0

```

imagem 4: Modo Sign-Magnitude

```

-5: 10 100
-4: 10 010
-3: 10 000
-2: 0 011
-1: 0 001
0: 0 000
1: 0 010
2: 0 100
3: 10 001
4: 10 011
5: 110 000
6: 110 010
7: 110 100
8: 1110 001
9: 1110 011
10: 11110 000
11: 11110 010
12: 11110 100
13: 111110 001
14: 111110 011
15: 1111110 000
16: 1111110 010
17: 1111110 100
18: 11111110 001
19: 11111110 011

```

imagem 5: Modo Interleaved

Parte 3 - Audio Encoding with Predictive Coding

O codec criado utiliza codificação preditiva como mecanismo principal de compressão. Em vez de codificar as amostras de áudio brutas, codifica as diferenças entre os valores previstos e os valores reais (resíduos). Esta abordagem é eficaz porque os sinais de áudio frequentemente apresentam forte correlação temporal, tornando as amostras próximas previsíveis a partir das anteriores.

Funcionamento Geral

O pipeline de compressão consiste em 3 passos fundamentais:

1. Fase de Previsão: O sistema emprega previsão baseada em polinómios de Taylor de grau variável para estimar as amostras seguintes com base nos valores anteriores. Para cada frame, o codec pode selecionar adaptativamente o grau de Taylor ótimo que minimiza a entropia, tornando a previsão mais precisa em diferentes tipos de conteúdo áudio, e em diferentes momentos de um mesmo ficheiro.

2. Cálculo de Resíduos: Após a previsão, o codec calcula os resíduos subtraindo os valores previstos das amostras reais. Estes resíduos tipicamente têm menor entropia que o sinal original, tornando-os mais compressíveis.

3. Codificação de Golomb: Os resíduos são comprimidos usando codificação de Golomb, uma técnica de codificação de entropia particularmente eficaz para codificar números inteiros pequenos centrados em zero. O parâmetro 'm' de Golomb é calculado dinamicamente para cada frame com base no valor absoluto médio dos resíduos, otimizando a taxa de compressão.

Modo Lossless vs Lossy

O codec suporta compressão com e sem perdas:

Modo Sem Perdas: Neste modo, os resíduos são codificados diretamente sem quantização, garantindo a reconstrução perfeita do sinal de áudio original. A taxa de compressão depende apenas da precisão da previsão e da eficiência da codificação de Golomb.

Modo Com Perdas: Este modo introduz quantização dos resíduos para atingir taxas de bits específicas. O factor de quantização (q_bits) é ajustado dinamicamente com base na taxa de bits alcançada em cada frame, permitindo que a taxa de bits do codec se aproxime o máximo possível da desejada.

Estrutura do Ficheiro

O ficheiro resultante da compressão tem a extensão .g7a. Internamente, começa por apresentar um cabeçalho que contém metadados transversais a todo o ficheiro:

- Número de canais
- Frequência de amostragem
- Tamanho dos frames
- Número total de samples
- Modo de codificar valores negativos para a codificação Golomb

Depois, o áudio é guardado em frames, com cada frame contendo:

- Um cabeçalho especificando o parâmetro de Golomb (m)
- O fator de quantização (q_bits)
- O grau de previsão de Taylor utilizado
- Os dados dos resíduos codificados

Esta abordagem baseada em frames permite a seleção adaptativa destes parâmetros e melhor compressão ao tirar partido das características locais do sinal.

Testes Realizados

Taxa de Compressão Lossless

Aplicando o codec no modo lossless aos 8 datasets utilizados, obtiveram-se as seguintes taxas de compressão:

CantinaBand	sample01	sample02	sample03	sample04	sample05	sample06	sample07
34%	30%	34%	40%	40%	50%	61%	20%

Isto resultou numa taxa média de compressão de 38.6%, ou uma média de 9.82 bits por amostra, no caso de áudio de 16 bits.

Taxa de Compressão Lossy

A compressão Lossy permitiu atingir níveis de compressão mais elevados, mas a custo de perda de qualidade.

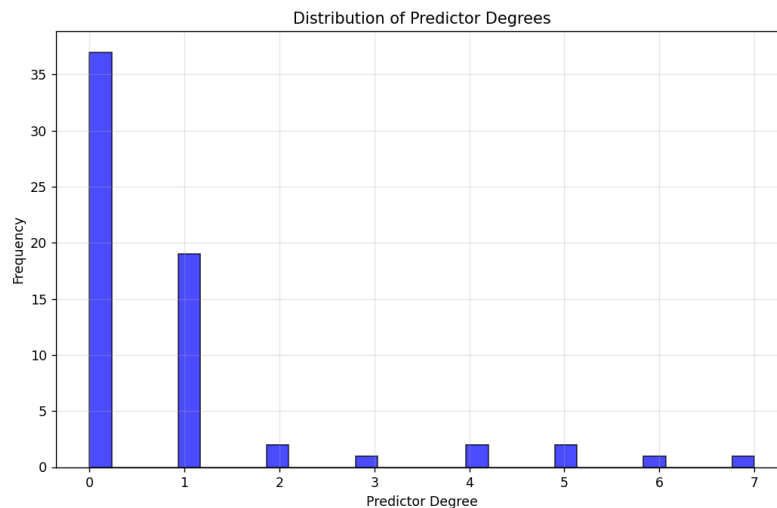
Como exemplo, fez-se uma análise subjetiva para diferentes níveis de compressão do ficheiro *sample05.wav*:

- Ficheiro não comprimido - bitrate médio de ~1400kbps
- Compressão lossless - bitrate médio de ~700kbps
- Compressão lossy para 400kbps - não se nota qualquer diferença
- Compressão lossy para 300kbps - é possível ouvir um ruído de fundo muito ligeiro
- Compressão lossy para 280kbps - nota-se alguma distorção
- Compressão lossy para 250kbps - o ruído sobrepõe-se completamente à música

Neste caso, a compressão Lossy permitiu uma redução de cerca de 70% no tamanho do ficheiro (de 1400 para 400kbps) sem perda de qualidade perceptível.

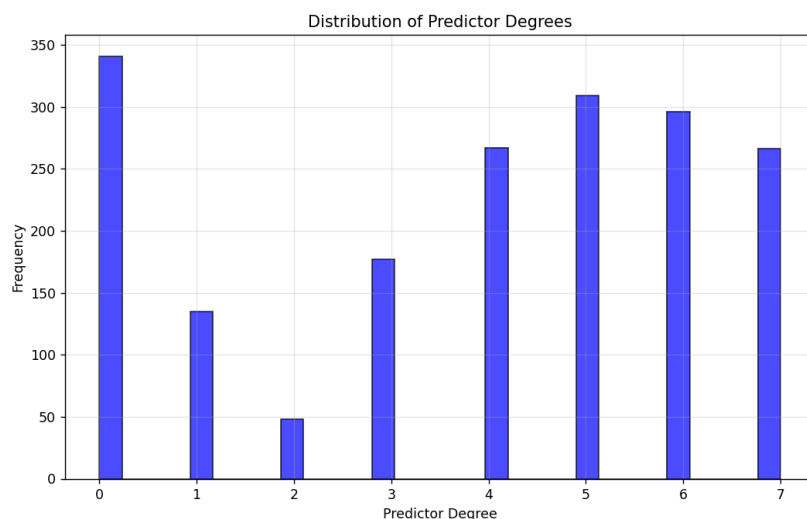
Grau Ótimo de Predição

Analizou-se o grau de previsão escolhido pelo encoder ao longo de todas as frames de todos os datasets usados. Os resultados são visíveis no histograma seguinte.



Vê-se que o preditor de grau 0, que corresponde a prever cada amostra como sendo igual à anterior, é o mais utilizado, seguido pelo preditor de grau 1, linear, sendo que os restantes são comparativamente pouco comuns.

No entanto, para alguns dos ficheiros comprimidos, a distribuição dos graus de preditor ótimos não segue necessariamente esta distribuição, como foi o caso do sample07.wav.



Parte 4 - Video Encoding with Predictive Coding