Codificação de Áudio e Vídeo  
1º Trabalho

Joel Pinheiro (65151)

Luis Assunção (42967)

*Resumo* – O objectivo deste primeiro trabalho foi entender a necessidade e funcionamento da codificação de informação. A codificação de informação consiste na modificação da informação para um fim em específico, seja transmissão de dados ou armazenamento. Para tal, é essêncial a existência de algoritmos de manipulação dos bits da informação a codificar/descodificar.

*Palavras Chave* – Codificação de informação; Códigos de Golomb; Manipulação de bits.

# I. Introdução

Neste primeiro trabalho temos como objetivo desenvolver um módulo que tenha a funcionalidade de ler e escrever um bit de cada vez, em sequência, como se fosse um stream; e também escrever conjuntos de bits de uma vez em sequência nesse mesmo ficheiro. Tivemos também que desenvolver um modulo que permita codificar e descodificar códigos Golomb Rice por meio do modulo mencionado anteriormente. Finalmente iremos analisar o desemprenho das nossas implementações, quer a nível temporal, quer a nível da taxa de compressão.

A codificação de Golomb é um sistema de compressão de dados inventado por Solomon W. Golomb em 1966 que pode ser utilizado na compressão de dados em substituição ao código de Huffmann, apresentando resultados ótimos para determinadas distribuições de probabilidade dos símbolos codificados. O sistema é baseado em codificação de entropia e tem como objetivo codificar números inteiros não negativos em que se verifica que a probabilidade de ocorrência é menor quanto maior for o número. A codificação de Rice foi inventada por Robert F. Rice e consiste numa adaptação da codificação de Golomb. Rice utilizou a codificação de Golomb e modificou a de forma a que um parâmetro (parâmetro ‘m’) passa-se a ser uma potência de 2, em vez de ser um número inteiro positivo. Este método irá simplificar a codificação e possivelmente acelerar a implementação, no entanto, dependendo dos casos, a taxa de compressão poderá ser menor.

# II. Códigos de Golomb

Para desenvolver um codificador de entropia, pode ser desenvolvido um módulo baseado no método de compressão de Golomb. Neste método é recebido um parâmetro de tuning que divide o valor recebido como entrada. O quociente desta divisão será representado de forma unária, enquanto que o resto será representado de forma binária com o número de bits truncado, de forma a fazer uma representação compacta.

Considerando , em que é o parâmetro de tuning previamente escolhido, e é o número que se pretende codificar. O Quociente é representado por , o resto é representado por e são determinados da seguinte forma:

Como , o tamanho de r é truncado da seguinte forma:

* Se , o tamanho é truncado em b-1 bits;
* Se , o tamanho é truncado em b bits.

Para obter a codificação, é necessário concatenar o valor de q, unário, com o valor de r, binário.

Para a descodificação, tem de ser feita uma análise bit a bit até se encontrado o primeiro bit com valor 1 (quociente) em seguida tem se ser determinado o valor do resto. Para obter o número de bits que o resto está codificado é necessário utilizar o algoritmo da codificação binária truncada, ou seja, se m é uma potência de base 2, então o resto é truncado em b bits, se não, é necessário efetuar o seguinte algoritmo:

* Calcular k, dada por , tal que
* Calcular u, através de
* Ler os primeiros k bits
* Se estes codificarem um valor menor que u, então a descodificação está finalizada
* Se não, é preciso ler mais um bit e subtrair o valor de u ao resto obtido de forma a obter o valor real do resto

Para finalizar, é obtido o valor recorrendo a:

# III. Extenção dos códigos de Golomb

# Uma vez que a codificação Golomb foi feita pensada apenas em inteiros positivos, torna-se necessário precaver a possibilidade da existencia de inteiros nao positivos durante a codificação e descodificação. Para tal, durante a codificação de um numero, primeiro verificamos se é positivo ou negativo.

# Caso seja positivo e não zero, no fim do código Golomb escrevemos '0', caso seja negativo escrevemos '1' respetivamente. Se o numero a codificar for '0', então nao escrevemos nenhum bit no final do código Golomb. É sempre codificado o valor absoluto do numero em questão, quer seja positivo ou negativo. Para a descodificação, verifica-se o bit final do código Golomb, e caso seja '1' o valor final será negativo, caso contrário é positivo.

# IV. BitStream

## BitStreamReader

A classe BitStreamReader e definida da seguinte maneira:

**class** BitStreamReader {  
**public**:  
 BitStreamReader(std::string filePath);  
 **int** readBit();  
**protected**:  
**private**:  
 **unsigned int** getBufferSize();  
 **void** refreshBufferContents();  
 std::ifstream fileRead;  
 **char**\* readBuffer;  
 **unsigned int** bufferSize;  
 **unsigned int** fileSize;  
 **unsigned int** fileReadPosition;  
 **int** bufferReadBitPosition;  
 **unsigned int** bufferReadBytePosition;  
};

O objetivo desta classe é fornecer uma abstração a manipulação de ficheiro bit a bit, que permita ler um bit de cada vez sequencialmente atraves do método readBit(). Este método retorna 0 ou 1, em conformidade com o estabelecido para o código binário natural.

## BitStreamWriter

A classe BitStreamWriter é definida da seguinte maneira:

**class** BitStreamWriter {  
**public**:  
 BitStreamWriter(std::string filePath);  
 **int** writeBit(**int** value);  
 **int** writeNBits(**int** value);  
 ~BitStreamWriter();  
 **void** paddAndCloseFile();  
**protected**:  
**private**:  
 **void** flushBufferContents();  
 std::ofstream fileWrite;  
 **char**\* writeBuffer;  
 **char** dataBeingProcessed;  
 **int** bufferWriteBitPosition;  
};

O objetivo desta classe e fornecer uma abstração a manipulação de ficheiro bit a bit, que permita escrever um bit ou um conjunto de bits de cada vez, em sequência, atraves do método writeBit() e writeNBits() respetivamente, em conformidade com o código binário natural. É também disponibilizado o método paddAndCloseFile(), que permite fechar o ficheiro aberto para escrita quando assim desejado.

# V. Golomb-Rice

## Golomb

A classe Golomb é definida da seguinte maneira:

**class** Golomb {  
**public**:  
 Golomb();  
 **void** golombEncode(**unsigned int** m, std::string n\_string, BitStreamWriter \*bitStreamWriter);  
 **int** golombDecode(**const int** m, BitStreamReader \*bitStreamReader, **int** &error);  
**protected**:  
**private**:  
 BitStreamReader \*bitStreamReader;  
 BitStreamWriter \*bitStreamWriter;  
};

Esta classe permite, através dos seus métodos públicos, codificar ou descodificar palavras(words com um tamanho máximo de 32 bits) de acordo com o enunciado por Solomon S Golomb. Para tal usamos os métodos golombEncode() e golombDecode() respetivamente. A classe Golomb tem relação de dependência para com as classes BitStreamWriter e BitStreamReader.

# VI. Execução

Nesta secção mostraremos como executar o nosso projecto.

O nosso projecto encontra-se estruturado da seguinte forma:

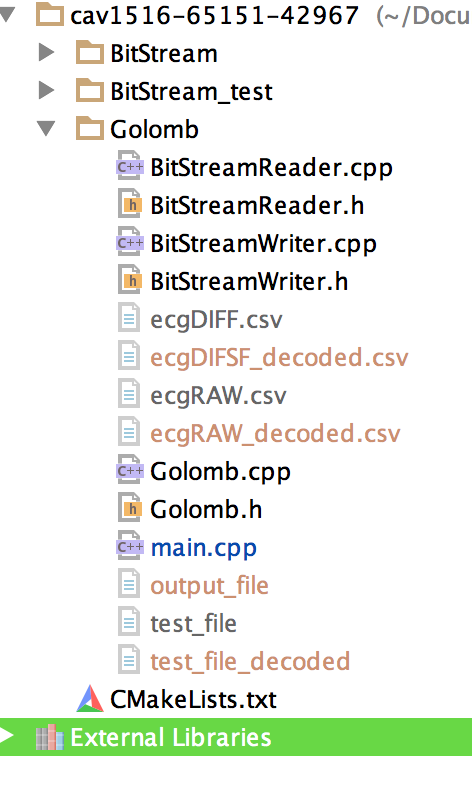


Figura 1 - Diretório do Projeto

# ../../../Desktop/Screen%20Shot%202015-10-12%20at%2000.55.58.png

Figura 2 - Execução do programa (configuração)

# VI. Análise de Resultados

Nesta secção iremos fazer uma análise aos resultados e tirar conclusões aos factos obtidos.

Considerando o ficheiro ecgRAW.csv (215 405 bytes).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Operação | M | Tempo (ms) | Tamanho | Taxa Compressão |
| Encode | 4 | 986 | 293245 | 136% |
| Encode | 8 | 975 | 236945 | 109% |
| Encode | 16 | 1028 | 236945 | 109% |
| Encode | 32 | 1570 | 215405 | 100% |
| Encode | 64 | 1321 | 215405 | 100% |
| Encode | 128 | 1353 | 242331 | 112% |
| Encode | 523 | 1489 | 296182 | 137% |

|  |
| --- |
| :: M ::8  Golomb Test File  Tempo de codificacao: 1646 milisegundos.  Tamanho do ficheiro codificado e: 293245  Tamanho do ficheiro original e: 215405  Taxa de compressão (%): 136  :: M ::16  Golomb Test File  Tempo de codificacao: 1632 milisegundos.  Tamanho do ficheiro codificado e: 236945  Tamanho do ficheiro original e: 215405  Taxa de compressão (%): 109  :: M ::32  Golomb Test File  Tempo de codificacao: 1242 milisegundos.  Tamanho do ficheiro codificado e: 215405  Tamanho do ficheiro original e: 215405  Taxa de compressão (%): 100  :: M ::64  Golomb Test File  Tempo de codificacao: 1198 milisegundos.  Tamanho do ficheiro codificado e: 215405  Tamanho do ficheiro original e: 215405  Taxa de compressão (%): 100  :: M ::128  Golomb Test File  Tempo de codificacao: 1226 milisegundos.  Tamanho do ficheiro codificado e: 242331  Tamanho do ficheiro original e: 215405  Taxa de compressão (%): 112  :: M ::256  Golomb Test File  Tempo de codificacao: 1205 milisegundos.  Tamanho do ficheiro codificado e: 269257  Tamanho do ficheiro original e: 215405  Taxa de compressão (%): 125 |

Figura 3 - Output Encode ecgRAW

|  |  |  |
| --- | --- | --- |
| Operação | M | Tempo (ms) |
| Decode | 4 | 43 |
| Decode | 8 | 38 |
| Decode | 16 | 42 |
| Decode | 32 | 50 |
| Decode | 64 | 53 |
| Decode | 128 | 66 |
| Decode | 523 | 71 |

Considerando o ficheiro ecgDIFF.csv (99 763 bytes)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Operação | M | Tempo (ms) | Tamanho | Taxa Compressão |
| Encode | 4 | 766 | 192625 | 193% |
| Encode | 8 | 604 | 130110 | 130% |
| Encode | 16 | 682 | 105149 | 105% |
| Encode | 32 | 656 | 99763 | 100% |
| Encode | 64 | 690 | 99763 | 100% |
| Encode | 128 | 761 | 112234 | 112% |
| Encode | 523 | 674 | 137175 | 137% |

|  |  |  |
| --- | --- | --- |
| Operação | M | Tempo (ms) |
| Decode | 4 | 33 |
| Decode | 8 | 32 |
| Decode | 16 | 26 |
| Decode | 32 | 23 |
| Decode | 64 | 22 |
| Decode | 128 | 24 |
| Decode | 523 | 48 |

# VI. Bibliogragia

1. <https://en.wikipedia.org/wiki/Truncated_binary_encoding>
2. <https://en.wikipedia.org/wiki/Golomb_coding>
3. <https://urchin.earth.li/~twic/Golomb-Rice_Coding.html>