# Trabalho Prático 1

Hugo Machado(A80362)
Mestrado Integrado em Engenharia de Telecomunicações e informática
Tecnologias e Serviços Multimédia
Bruno Dias
Universidade do Minho

April 6, 2020

# 1 Estratégias Escolhidas

## 1.1 Compressor

Este código está dividido em 5 partes diferentes:

- Contagem de ocorrências de cada símbolo num bloco;
- Redução do número de ocorrências para caber num *byte*;
- Ordenação da lista de ocorrências;
- Calculo da tabela de Shannon-Fano;
- Construção do bloco compresso;

### Contagem de ocorrências

Nesta parte é lido um bloco de 2mb do ficheiro, e são contados o número de ocorrências de cada carácter e ao mesmo tempo verificamos qual o carácter com maior ocorrências.
Sabendo o maior número de ocorrências, é possível calcular a melhor divisão para a seguinte redução.

### Redução do número de ocorrência

Aqui é inicializado um *array* para a ordenação das ocorrências, a redução de cada ocorrência, o calculo do novo numero total de ocorrências e preenchemos o *header* do bloco com as ocorrências de cada carater. Para optimizar todo este processo englobei todos este cálculos num só ciclo e evitei a redução quando a ocorrência é zero.

#### Ordenação

Na ordenação usei o algoritmo *Bubble-Sort*, para optimizar este processo utilizei a operação binária *XOR*. A utilização desta operação faz com que não seja necessário a utilização de uma variável auxiliar.

#### Tabela de Shannon-Fano

Na calculo da tabela de Shannon-Fano, é usado recursividade. E em todos os cálculos são feitos com inteiros(ocorrências), desta forma por não trabalhar com valores decimais o calculo é mais eficiente.

#### Construção do Bloco Compresso

Na construção do bloco compresso, utilizamos o carácter do bloco como índice da tabela, e escreve-mos o respectivo código *bit-a-bit*. Aqui são utilizadas operações binárias para tornar este processo o mais eficiente possível.

### 1.2 Descompressor

Este código está dividido em 5 partes diferentes:

- Leitura dos valores do *header*;
- Ordenação da lista de ocorrências;
- Calculo da tabela de Shannon-Fano;
- Preenchimento da árvore binária;
- Construção do bloco original;

#### Leitura do *header*

Nesta secção são lidas todas as ocorrências de cada carácter, é inicializado um *array* para a ordenação das ocorrências, o calculo do novo número total de ocorrências, tudo no mesmo ciclo. Depois com operações binárias são lidos os valores, número de caracteres do bloco e o tamanho do bloco.

#### Ordenação e Tabela de Shannon-Fano

Estes algoritmos são exactamente iguais aos do código compressor.

#### Árvore binária

Na descompressão são utilizadas árvores binárias, para que a associação entre código e carácter seja mais eficiente.

#### Construção do Bloco Original

Por fim, nesta parte lê-se o bloco *bit-a-bit*, de seguida por cada *bit* iteramos a árvore binária, até chegar ao respectivo carácter.

### 1.3 Outras optimizações

- Apenas foi utilizado a memoria necessária, evitando assim desperdício de memória;
- *Flag* "-O3" activa múltiplas optimizações que ajudam na compilação de um executável mais optimizado;
- *Flag* "-march=native" compila um executável mais optimizada para a respectiva maquina.
- Foram utilizadas operações4 binárias sempre que possível.

## 2 Testes e Resultados

Para testar o programas feitos neste trabalho prático, foram fornecidos vários ficheiros para teste. E para ter noção do desempenho do programa, estão referidos os tempos e tamanhos do programa desenvolvido, e também os mesmo do Zip em modo normal e rápido.

### Compressão

| Ficheiro | Compressor | Zip(Normal) | Zip(Optimizado) |
|---|---|---|---|
| texto.txt - 5.34mb | 217ms - 3.23mb | 746ms - 2.03mb | 200ms - 2.39mb |
| music.mp3 - 20,08mb | 703ms - 20,08mb | 1393ms - 20.00mb | 1331ms - 20.01mb |
| music.flac- 50,90mb | 1622ms - 51,08mb | 2885ms - 50.91mb | 2714ms - 50.91mb |
| music.wav - 88.54mb | 3442ms - 86.31mb | 6079ms - 85.89mb | 5822ms - 85.97mb |

### Descompressão

| Ficheiro | Descompressor | Unzip(Normal) | Unzip(Optimizado) |
|---|---|---|---|
| texto.txt | 177ms | 88ms | 94ms |
| music.mp3 | 523ms | 288ms | 288ms |
| music.flac | 1060ms | 541ms | 554ms |
| music.wav | 2872ms | 1348ms | 1384ms |

*Unzip(Optimizado): este é o resultado da descompressão dos ficheiro compressos através do Zip(optimizado).

# 3 Limitações

Apesar de todas a optimizações efectuadas, a ferramenta continua a ter lugar para melhores optimizações, principalmente na escrita do bloco compresso.
Depois em relação a compressão a ferramenta está limitada ao algoritmo de Shannon-Fano, podemos ainda utilizar um mecanismo *Run-Length Encoding*(RLE), para reduzir a concentração espacial, mas não seria suficiente para obter melhores resultados que as actuais ferramenta de compressão.
Por fim, ainda existem alguns mecanismos que podem ser problemáticos, como a recursividade.

# References


[1] Bruno Dias, Bibliografia  Material, Slides do Trabalho pratico.

[2] C C++ code optimization,
https://www.thegeekstuff.com/2015/01/c-cpp-code-optimization/

[3] C for speed,
https://www.atarimagazines.com/startv5n6/c$_{f}or_{s}peed.html$

[4] How to optimize C,
https://medium.com/@aka.rider/how-to-optimize-c-and-c-code-in-2018-bd4f90a72c2b

[5] 10 Simple tricks to optimize your C code in small embedded systems,
https://www.embedded.com/10-simple-tricks-to-optimize-your-c-code-in-small-embedded-sy