Trabalho Prático 2

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

May 4, 2020

1 Estratégias Escolhidas

1.1 Compressor

Este código está dividido em 9 partes diferentes:

- Leitura de um bloco;
- Deteção automática *RLE*;
- Deteção automática *LZW*;
- Contagem de ocorrências de cada símbolo num bloco;
- Redução do tamanho das ocorrências;
- Ordenação da lista de ocorrências;
- Calculo da tabela de *Shannon-Fano*;
- Construção do bloco compresso;
- Escrita do bloco.

Leitura de um bloco

O primeiro processo deste algoritmo é a leitura de um bloco de 2MB do ficheiro indicado.

Deteção automática ***RLE***

Este mecanismo, *RLE = Run Length Encoding*, é um simples algoritmo de compressão que elimina a concentração espacial de um dado símbolo.
Este funciona de forma automática, ou seja, se a dado momento o contador de caracteres do bloco compresso for superior ao contador do bloco original, este mecanismo é desativado para o resto do ficheiro.

Deteção automática *LZW*

O mecanismo, *LZW = Lempel–Ziv–Welch*, é a principal característica deste segundo trabalho prático. Este trata-se de um mecanismo de codificação por dicionário. O seu funcionamento trata-se de construir padrões através das sequências dos múltiplos símbolos, e para cada padrão é atribuído um código incrementativo.
Para a otimização deste mecanismo utilizei a sugestão do professor, usar uma matriz de forma a servir como se fosse um mapa de memoria, e em cada posição temos a próxima linha, onde teremos as próximas linhas conforme o seguinte símbolo. Apenas com uma pequena alteração, em cada posição tenho código do padrão até aquele ponto e a linha seguinte.

Contagem de ocorrências

Nesta parte é contado o número de ocorrências de cada simbolo 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 7 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;
- Deteção automática *LZW*;
- Deteção automática *RLE*;

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

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

Deteção automática *LZW*

Após a construção do bloco, verificamos se ele tem 2MB(Tamanho Original), se sim é porque não existe compressão *LZW* e a partir deste ponto a compressão *LZW* é desativada, se não é porque existe compressão LZW, e é executado o processo de descompressão *LZW*.
No processo de descompressão, apenas é utilizado um *Array de Strings* e o código de cada padrão é o índice da *String* que contém o padrão.

Deteção automática *RLE*

Tal como a deteção *LZW*, a deteção *RLE* segue o mesmo principio.

1.3 Outras optimizações

- Apenas foi utilizado a memoria necessária, evitando assim o 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 operações 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	170ms - 2.92mb	338ms - 2.03mb	99ms - 2.39mb
music.mp3 - 20,08mb	380ms - 20,08mb	650ms - 20.00mb	580ms - 20.01mb
music.flac- 50,90mb	895ms - 51,08mb	1285ms - 50.91mb	1199ms - 50.91mb
music.wav - 88.54mb	1786ms - 86.31mb	2750ms - 85.89mb	2561ms - 85.97mb

Descompressão

Ficheiro	Descompressor	Unzip(Normal)	Unzip(Optimizado)
texto.txt	96ms	53ms	50ms
music.mp3	250ms	144ms	137ms
music.flac	510ms	261ms	254ms
music.wav	1337ms	591ms	591ms

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

Modo *Verbose*, do texto.txt

```
--------------------Bloco nr 1--------------------

Reducao RLE = 2.221%
Tempo de compressao RLE = 4.410ms
Reducao LZW = 42.531%
Tempo de compressao LZW = 39.090ms
Tempo de leitura e contagem de ocorrencias = 1.412ms
Tempo de reducao das ocurrencias = 0.006ms

Entropia do bloco 8.251 bits/simbolo
Tempo de ordenacao = 0.062ms
Tempo de Shannon-Fano = 0.009ms
Tempo de Construcao e escrita do payload = 25.697ms
Comprimento medio do bloco = 7.783
Mostrar Tabela de Shannon-Fano?
(s)im ou (n)ao
n

(Pressione enter para continuar)

--------------------Bloco nr 2--------------------

Reducao RLE = 2.097%
Tempo de compressao RLE = 5.108ms
Reducao LZW = 42.941%
Tempo de compressao LZW = 33.685ms
Tempo de leitura e contagem de ocorrencias = 2.129ms
Tempo de reducao das ocurrencias = 0.011ms

Entropia do bloco 8.268 bits/simbolo
Tempo de ordenacao = 0.095ms
Tempo de Shannon-Fano = 0.012ms
Tempo de Construcao e escrita do payload = 24.999ms
Comprimento medio do bloco = 7.783
Mostrar Tabela de Shannon-Fano?
(s)im ou (n)ao
n

(Pressione enter para continuar)

--------------------Bloco nr 3--------------------

Reducao RLE = 1.000%
Tempo de compressao RLE = 2.955ms
Reducao LZW = 42.531%
Tempo de compressao LZW = 29.719ms
Tempo de leitura e contagem de ocorrencias = 0.988ms
Tempo de reducao das ocurrencias = 0.007ms

Entropia do bloco 8.240 bits/simbolo
Tempo de ordenacao = 0.067ms
Tempo de Shannon-Fano = 0.009ms
Tempo de Construcao e escrita do payload = 18.952ms
Comprimento medio do bloco = 7.779
Mostrar Tabela de Shannon-Fano?
(s)im ou (n)ao
n

(Pressione enter para continuar)

------------------------Fim-------------------------

Reducao de 45.272%
Comprimento medio do ficheiro = 7.782
Entropia do ficheiro = 8.253 bits/simbolo
```

Figure 1: Modo *Verbose*

3 Limitações

Apesar de todas a optimizações efectuadas, a ferramenta continua a ter lugar para melhores optimizações, principalmente na escrita do bloco através codificação estatística.
Por fim, ainda existem alguns mecanismos que podem ser problemáticos, como a recursividade.

4 Manual

Para aceder ao manual de instruções basta correr o programa sem argumentos, `./covid`, e o seguinte manual aparecerá.

```
Copyright (c) Hugo Machado 2019-2020 TSM
Covid 19.0  Usage(linux only):
(for a better and faster usage, compile with the next flags)

->   gcc -o covid covid.c -O3 -march=native
->   ./covid [-options] [file to compress/decompress] [name of result file]

Avaible options:
-l  verbose/debug mode(only compress)
-c  compress mode
-d  decompress mode
```

Figure 2: Manual

References

[1] Bruno Dias, Bibliografia e 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`$_{for}$$_{speed.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`