Compressor Shannon Fano

Rui Pedro Azevedo Oliveira

January 7, 2014

Contents

Abstract

O Objetivo deste trabalho era a criação de uma ferramenta de compressão utilizando o algoritmo de *Shanno Fano* e *Run-length encoding*.

Neste relatório poderám ser encontrado três capitulos: "Código", "Testes" e "Conclusões". No capitulo Código irá ser falada a estrutura do código e alguns dados tecnicos sobre a implementação do algoritmo. Em Testes irám ser descritos alguns testes realizados a esta ferramenta e finalmente em Conclusões serám descritas todos as conclusões que se obteve com este trabalho.

Chapter 1

Código

Explicação da estrutura do código

A linguagem usada para a realização do trabalho foi *C++*.

O código foi editado com o *Gedit*, e compilado por *G++*.

1.1 Estrutura do Código

Os ficheiros com o código fonte estão na pasta *"source"*, nesta pasta podemos encontrar os seguintes ficheiros de código:

- *"main.cpp"*
- *"decoder.cpp"* Motor de descompressão
- *"encoder.cpp"* Motor de compressão
- Sub pasta *"struct"* com a estrutura de dados:
  - *"shan_table.class.cpp"* Tabela Shannon Fano
  - *"shan_row.class.cpp"* Linhas da tabela Shannon Fano

A extensão usada para o código foi *.cpp* e *.h* para o cabeçalho, note-se que todos os fichiros *.cpp* tem um ficheiro correspondente *.h*

O desenvolvimento do código foi feito de forma modelar, assim caso seja necessário mudar alguma funcionalidade, para por exemplo otimizar, é mais fácil.

1.2 Motor de Compressão

O Motor de compressão está no ficheiro *"encoder.cpp"*. O seu funcinamente pode ser divido em duas grandes partes: **ler amostra** e **escrever compressão**, estas duas ações está respectivamente nas funções: *startSample* e *startWriting*.

As amostras são tiradas no ficheiro todo, uma vez que a maior parte do tempo de execução estão na descompressão (ver 3.3, e iria compremeter a qualidade das amostras. O tamanho das palavras são: 1 *Byte*.

Após ter a lista de frequências preenchida irá ser aplicado o algoritmo de Shannon fano, para mais pormenores consultar 1.4.

1.2.1 Escrita da Tabela de Frequências

O ficheiro comprimido está agora pronto para começar a ser escrito, a primeira coisa a escrever é a tabela de frequências: existem $2^8 + 1 = 257$ frequências diferentes e cada frequência é um *unsigned long*.

A forma encontrada para escrever todo esta informação foi escrever um *array* de *unsigned long* com comprimento 257.

1.2.2 Escrita do Ficheiro

Os dados relativos ao ficheiro são agora escritos. Para isso lê-se o ficheiro original e para cada *Byte* é acedida a tabela *Shannon Fanno* e é escrito o numero variável de *bits* correspondentes ao *Byte* original.

O ultimo *Byte* do ficheiro é um palavra especial que sinaliza o fim do ficheiro.

Para a escrita e leitura é usada um buffer de tamanho 4048 *bytes*.

1.3 Motor de Descompressão

O Motor de Descompressão está no ficheiro *"decoder.cpp"*. O seu funcionamento tal como o motor de compressão está divido em duas grandes parte: **ler Tabela** e **escrever descompressão**, estas duas ações está respectivamente nas funções: *startReading* e *startWriting*.

1.3.1 Leitura da Tabela de Frequências

A leitura da tabela de Frequências exatamente a operação inversa da escrita no motor de compressão. Lê-se um *array* de *unsigned long* com comprimento 257.

Onde a frequência de um determinado *char* com valor N é: o valor do array na posição N.

A tabela de *Shannon Fanno* pode agora ser criada.

1.3.2 Escrita do Ficheiro

Para a escrita do ficheiro original, é lido o ficheiro comprimido *bit* por *bit* e é acedida a tabela de *Shannon Fanno*.

Numa primeira fase o programa utilizava a função $voidgetCharByBitsTable(chara, intbitIndex, u$ $index)$ que para encontrar uma palavra percorria toda a tabela de palavras, tempo linear.

Como otmização criou-se $voidgetCharByBitsGraph(chara, intbitIndex, unsignedint*$ $index)$ que utiliza um grafo para encontrar as palavras em tempo constante. Para mais informações sobre o grafo consultar 1.4.

1.4 Tabela Shannon Fano

A criação da tabela de *Shannon Fanno* é feito na class *ShanTable* no ficheiro *Shan_ table.class.cpp*. Aqui são guardados dos dados Relativos á tabela: são eles as frequências, e o conjunto de bits correspondentes a cada palavra - ver 1.4.1

Quando este é executado o $voidShanTable :: createShanTable()$ a class cria a tabela em duas formas: Forma de Tabela e Forma de Grafo.

Forma de tabela para acesso direto quando se está a realizar a compressão - são copiados blocos de *bits* de uma vez aproveitando a mecânica de um *byte* completo, e é acedido em tempo constante.

Forma de Grafo para acesso direto quando se está a realizar a descompressão - mais fácil para acesso *bit* por *bit*, e tempo constante para procura.

Esta classe tem a particularidade de poder ter caracteres especiais para fins heurísticos. Como é o caso do caractere que simboliza o fim do ficheiro, outra possível funcionalidade desta particularidade pode ser vista em 3.4

1.4.1 Classe ShanRow

Esta classe tem por fim guardar e gerir um número variável de *bits*. Guarda os Bits numa lista de *unsingned char* com tamanho variável, se precisar reinstancia com o dobro do tamanho.

Chapter 2

Testes

Diferentes testes executados ao código.

Máquina usada:

Sistema Operativo: Fedora 19.

CPU: Intel Core i7-4700MQ CPU 2.40GHz x8.

GPU: Intel Haswell Mobile.

Memoria: 7.7GiB.

Para a medição do tempo foi usado o comando *time* e foi tido em conta o tempo real, e a comparação de ficheiros foi usada o comando *md5sum*.

2.1 Ficheiro ProsperDataExport_xml com 3.1GB

O ficheiro pode ser encontrado no endereço:
`https://services.prosper.com/DataExport/ProsperDataExport_xml.zip`

2.1.1 Com RLE

Tempo de Compressão: 1m51.789s

Tempo de Descompressão:3m38.293s.

Tamanho comprimido: 3.0GB.

Comparação de ficheiros, usando md5sum:

Original: b901234ffc0f1e1a3826271b493c9831.

Final: b901234ffc0f1e1a3826271b493c9831

2.1.2 Sem RLE

Tempo de Compressão: 1m0.772s

Tempo de Descompressão: 2m35.924s

Tamanho comprimido: 2.2GB .

Comparação de ficheiros, usando md5sum:

Original: b901234ffc0f1e1a3826271b493c9831.

Final: b901234ffc0f1e1a3826271b493c9831

2.2 Ficheiro 201061200010_1 com 22MB

O ficheiro pode ser encontrado no endereço:
`http://www.nbb.be/DOC/BA/PDF7MB/2010/201061200010_1.PDF`

2.2.1 Com RLE

Tempo de Compressão: 0m0.727s

Tempo de Descompressão: 0m1.686s

Tamanho comprimido: 27 MB.

Comparação de ficheiros, usando md5sum:

Original: 70f7ad2738d496aef091083f1b78ad01.

Final: 70f7ad2738d496aef091083f1b78ad01

2.2.2 Sem RLE

Tempo de Compressão: 0m0.475s

Tempo de Descompressão: 0m1.610s

Tamanho comprimido: 22.5MB .

Comparação de ficheiros, usando md5sum:

Original: 70f7ad2738d496aef091083f1b78ad01.

Final: 70f7ad2738d496aef091083f1b78ad01

2.3 Ficheiro com 2.5MB extensão .txt

Este ficheiro pode ser encontrado na pasta *ficheiros_ teste*

2.3.1 Com RLE

Tempo de Compressão: 0m0.054s

Tempo de Descompressão: 0m0.053s.

Tamanho comprimido: 1MB.

Comparação de ficheiros, usando md5sum:

Original: 8e064e2b7a73deee4cce28aafe90e998.

Final: 8e064e2b7a73deee4cce28aafe90e998

2.3.2 Sem RLE

Tempo de Compressão: 0m0.043s.

Tempo de Descompressão: 0m0.031s.

Tamanho comprimido: 635.2 Kb.

Comparação de ficheiros, usando md5sum:

Original: 8e064e2b7a73deee4cce28aafe90e998.

Final: 8e064e2b7a73deee4cce28aafe90e998

2.4 Ficheiro com 365.8kB extensão .jpg

Este ficheiro pode ser encontrado na pasta *ficheiros_ teste*

2.4.1 Com RLE

Tempo de Compressão: 0m0.016s.

Tempo de Descompressão: 0m0.030s.

Tamanho comprimido: 450.1kB.

Comparação de ficheiros, usando md5sum:

Original: a6383fe291c909235c766599821baa35.

Final: a6383fe291c909235c766599821baa35

2.4.2 Sem RLE

Tempo de Compressão: 0m0.010s.

Tempo de Descompressão: 0m0.029s

Tamanho comprimido: 369.9 kB.

Comparação de ficheiros, usando md5sum:

Original: a6383fe291c909235c766599821baa35.

Final: a6383fe291c909235c766599821baa35

2.5 Ficheiro com 458.0MB extensão .zip

Este ficheiro não pode ser encontrado na *Internet*.

2.5.1 Com RLE

Tempo de Compressão: 0m16.498s.

Tempo de Descompressão: 0m36.252s.

Tamanho comprimido: 579.4 MB.

Comparação de ficheiros, usando md5sum:

Original: 1fabcc3f772ba8b2fc194d6e0449da17.

Final: 1fabcc3f772ba8b2fc194d6e0449da17.

2.5.2 Sem RLE

Tempo de Compressão: 0m9.944s.

Tempo de Descompressão: 0m28.010s.

Tamanho comprimido: 468.0MB.

Comparação de ficheiros, usando md5sum:

Original: 1fabcc3f772ba8b2fc194d6e0449da17.

Final: 1fabcc3f772ba8b2fc194d6e0449da17

Chapter 3

Conclusões

3.1 Níveis de Compressão

Os níveis de compressão em ficheiros já comprimidos por natureza, como é o caso dos *.jpg, .rar e .zip* não são muito bons, pois já não existe muito mais para comprimir. Mas em ficheiros de texto o nivel de compressão é melhor, passando a ter menos de metade do tamanho original quer com RLE ou sem RLE.
O que sugere que nem todos os ficheiros podem ser comprimidos com sucesso e se mesmo assim quisermos comprimir irá ser criado um ficheiro maior que o original.

3.2 Uso ou não de RLE

Em teoria o RLE devia ser vantajoso nos ficheiros com muitos caracteres repetidos, como é o caso do ficheiro de texto com 2.5MB (tem bastantes espaços seguidos em vários sítios).

Mas nem neste ficheiro o uso de RLE ajudou a comprimir, portanto é provável que para os ficheiros quotidianos o uso de RLE traga piores níveis de compressão.

3.3 Tempos de execução

O tempo de Descompressão é, em média, o dobro do tempo de compressão, o que é provável que se deva ao facto de ter que correr o ficheiro *bit*

por *bit*.

3.4 Possiveis Otimizações

Algumas possiveis otimizações para que a aplicação conseguisse diminuir os tempos de execução seriam:

- Correr o ficheiro uma única vez durante a descompressão, colocando ao longo do ficheiro as alterações na tabela.
- Usar RLE apenas quando fossem detectados um certo número de palavras repetidas.
- Deixar de correr ficheiro *bit* por *bit* na descompressão, passando a comparar bloco por bloco.

3.5 Conclusão

O algoritmo de Shannon Fano não é complicado, mas a forma como os ficheiros estão organizados e são lidos/escritos torna a sua implementação um pouco difícil, o natural é ler e escrever 8 *bits* de cada vez, escrever e ler um número variável de *bits* é um pouco complicado.

Os níveis de compressão irão depender do ficheiro a ser comprimido, podem ser bons ou maus, dependendo se já está comprimido ou não.