Universidade do Porto

Faculdade de Engenharia

FEUP

Relatório Projecto 2

T2: Data Compression

Concepção e Análise de Algoritmos 2012/13

Turma 2MIEIC2 - Grupo D

Elementos:
Jorge Reis – 200800544 – ei08053@fe.up.pt
Miao Sun – 200803743 – ei08162@fe.up.pt
Vítor Castro – 200900648 – ei09131@fe.up.pt

24 de Maio de 2013

Índice de Matérias

Introdução

Foi proposto um projecto, no contexto da disciplina de Conceção e Análise de Algoritmos, que consiste na elaboração de uma aplicação que permita a compressão de ficheiros de texto (sem perdas de informação), e posterior análise das suas eficiências temporais e espaciais. Para tal, foram implementados os algoritmos *Keyword Encoding*, *Huffman codes* e *LZW (Lempel-Ziv-Welch)*, e respectivos descodificadores. São algoritmos que utilizam técnicas diferentes de compressão, como se verá mais à frente neste relatório.

1. Descrição do Problema

Em Teoria da Informação, compressão de dados envolve codificar informação utilizando menos bits do que o necessário para representá-la originalmente; este processo poderá acontecer com ou sem perda de conteúdo. Apesar da relativa abundância de espaço para armazenamento digital, ou de largura de banda para comunicação, compressão de dados continua a ser fundamental para diversas aplicações, especialmente em aplicações de tempo real, em que restrições temporais impõem-se à sequência dos processos de codificação, transferência e descodificação de dados. (*in: enunciado*)

Neste trabalho, pretendia-se implementar diferentes algoritmos com vista à compressão de ficheiros de texto, sem perdas de informação. Outro objectivo principal era analisar o desempenho de cada algoritmo, em função tanto do tempo que cada um demora a realizar o processo de compressão e descompressão, como da taxa de compressão resultante.

Mediante discussão com o professor da unidade curricular, ficou definida a implementação dos algoritmos: *Keyword Encoding*, *Huffman codes* e *LZW (Lempel-Ziv-Welch)*. Resolveu-se não implementar o método *Run-length encoding (RLE)*, visto que para ficheiros de texto normais prevê-se que teria uma eficiência de compressão quase nula.

2. Implementação

2.1. Algoritmos Implementados

O trabalho baseia se principalmente em três algoritmos: KeywordEncoding, Huffman e Lempel-Ziv-Welch (LZW).

Os três algoritmos foram implementados com o objectivo de comprimir texto e descomprimir sem perdas.

Vamos de seguida explicar um pouco estes algoritmos:

2.1.1. Keyword Encoding

O algoritmo Keyword Encoding é muito simples, sendo mais utilizado na compressão de ficheiros. O algoritmo apenas substitui as palavras mais comuns, que aparecem mais vezes no ficheiro, por caracteres especiais ou sequências especiais de caracteres, assim comprimido o tamanho do ficheiro.

Na implementação do código, após a leitura do ficheiro, são identificadas todas as palavras do texto, e definidas as suas frequências de aparecimento no mesmo. De seguida, as palavras mais comuns são codificadas em caracteres especiais, sendo o resultado gravado no ficheiro "comprimido". É também guardado um ficheiro auxiliar com as palavras que foram codificadas e respectivos caracteres especiais. Assim, é possível, com base neste ficheiro, repor o ficheiro original, "descomprimido".

2.1.2. Huffman Codes

A codificação de Huffman é um método de compressão que utiliza as frequências de ocorrência dos caracteres no ficheiro a ser comprimido para determinar códigos binários de tamanho variável para cada caracter. O algoritmo foi desenvolvido em 1952 por David A. Huffman que era, na época, estudante de doutoramento no MIT.

Assim, a ideia consiste em associar números binários com menos bits aos caracteres mais usados num texto. O algoritmo de Huffman que será implementado para codificar um arquivo consiste em três fases:

1. Na primeira fase, calcula-se a frequência com que ocorre cada carácter no texto.

2. A segunda fase do algoritmo consiste em construir uma arvore binária baseada na frequência de uso destas letras de modo que as mais frequentes apareçam mais perto da raiz que as menos frequentes. Assim, as folhas da árvore representam os caracteres.

3. Finalmente na terceira fase a árvore de Huffman será usada para codificar e descodificar o texto.

Este algoritmo apresenta uma complexidade $O(M + N \log N)$, em que M representa o comprimento do texto, e N o número de caracteres distintos.

2.1.3. LZW (Lempel-Ziv-Welch)

O LZW (Lempel-Ziv-Welch) é um algoritmo de compressão de dados, derivado do algoritmo LZ78, baseado na localização e no registo dos padrões de uma estrutura. Foi desenvolvido e patenteado em 1984 por Terry Welch.

Para este efeito existe um dicionário que é inicializado com todos os símbolos do alfabeto (ao se usar codificação ASCII são 256 símbolos, de 0 a 255).

A entrada é lida e acumulada numa cadeia de caracteres que vamos chamar de I.

Sempre que a sequência contida em I não estiver presente no dicionário emitimos o código correspondente à versão anterior de I (ou seja, I sem o último carácter) e adicionamos I ao dicionário. I volta a ser inicializado com o último carácter lido (o seu último carácter) e o processo continua até que não haja mais caracteres na entrada.

Na implementação do algoritmo LZW de modo a torna-lo mais eficiente, foram usadas duas tabelas de Hash para guardar o dicionário (Palavra>código ; código>Palavra), uma necessária para compressão e a outra para descompressão. Como as consultas e a inserções na tabela de Hash tem um peso O(1), a eficiência do algoritmo é da ordem de O(n).

No quadro seguinte mostrámos os passos da compressão da cadeia "A_ASA_DA_CASA" usando o método de LZW.

A coluna I representa a cadeia lida na entrada desde que a última saída foi emitida.

A coluna no dic? indica se a sequência em I está presente no dicionário.

A coluna nova entrada mostra o que será inserido no dicionário (como o dicionário já contém os 256 símbolos do código ASCII. É impraticável mostrar todo seu conteúdo, por isso listamos apenas as novas entradas.

Por fim a coluna saída mostra o código que será emitido na saída do programa, junto com o que ele representa no dicionário (entre parênteses).

I	no dic?	nova entrada	saída
A	S		
A_	N	256:A_	65 (A)
_	S		
_A	N	257:_A	95 (_)
A	S		
AS	N	258:AS	65 (A)
S	S		
SA	N	259:SA	83 (S)
A	S		
A_	S		
A_D	N	260:A_D	256 (A_)
D	S		
DA	N	261:DA	68 (D)
A	S		

A_	S		
A_C	N	262:A_C	256 (A_)
C	S		
CA	N	263:CA	67 (C)
A	S		
AS	S		
ASA	N	264:ASA	258 (AS)
A	S		
-	-	-	65 (A)

O tamanho final é de 10 códigos, sendo 7 representados por 8 bits cada um e 3 código representados por 9 bits. Nesse caso temos na saída 83 bits, comparados com os 104 originais. O método LZW é lento a fazer efeito visto que o dicionário está inicialmente preenchido com todos os símbolos do alfabeto. Contudo é mais simples de ser implementado e gera bons resultados mesmo com cadeias de caracteres mais longas.

2.2. Análise Comparativa

Keyword Encoding

Nome do Ficheiro de Entrada	Tamanho do Ficheiro de Entrada	Nome do Ficheiro de Saída	Tamanho do Ficheiro de Saída	Tempo de Compressão	Taxa de Compressão
Computer Science.txt	11,8 KB	Computer Science.ke	11,03 KB	0,22 seg	6,53%
100kb.txt	100 KB	100kb.ke	98,3 KB	0,94 seg	1,7%
sherlock.txt	6,18 MB	sherlock.ke	5,96 MB	820 seg	3,56%

Huffman Codes

Nome do Ficheiro de Entrada	Tamanho do Ficheiro de Entrada	Nome do Ficheiro de Saída	Tamanho do Ficheiro de Saída	Tempo de Compressão	Taxa de Compressão
Computer Science.txt	11,8 KB	Computer Science.hf	6,54 KB	0,01 seg	44%
100kb.txt	100 KB	100kb.hf	56,7 KB	0,035 seg	43,3%
sherlock.txt	6,18 MB	sherlock.hf	3,51 MB	1,83 seg	43,2%

LZW (Lempel-Ziv-Welch)

Nome do Ficheiro de Entrada	Tamanho do Ficheiro de Entrada	Nome do Ficheiro de Saída	Tamanho do Ficheiro de Saída	Tempo de Compressão	Taxa de Compressão
Computer Science.txt	11,8kb	Computer Science.lzw	8,08 KB	3,7 seg	31.53%
100kb.txt	100kb	100kb.lzw	45,6 KB	17 seg	55%
sherlock.txt	6,18 MB	sherlock.lzw	1,67 MB	10,27 min	72.98%

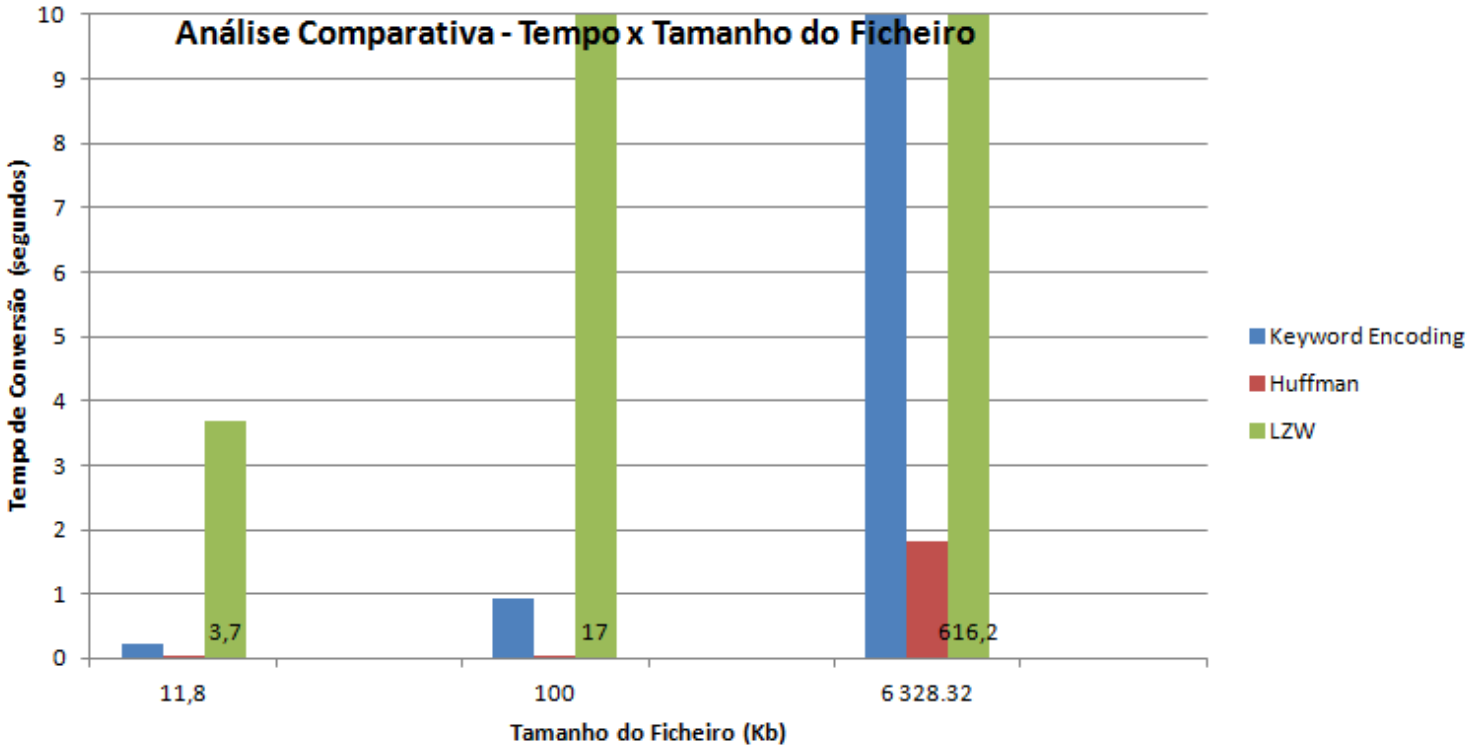

Figura 1 – Gráfico Comparativo Tempo x Tamanho Ficheiro

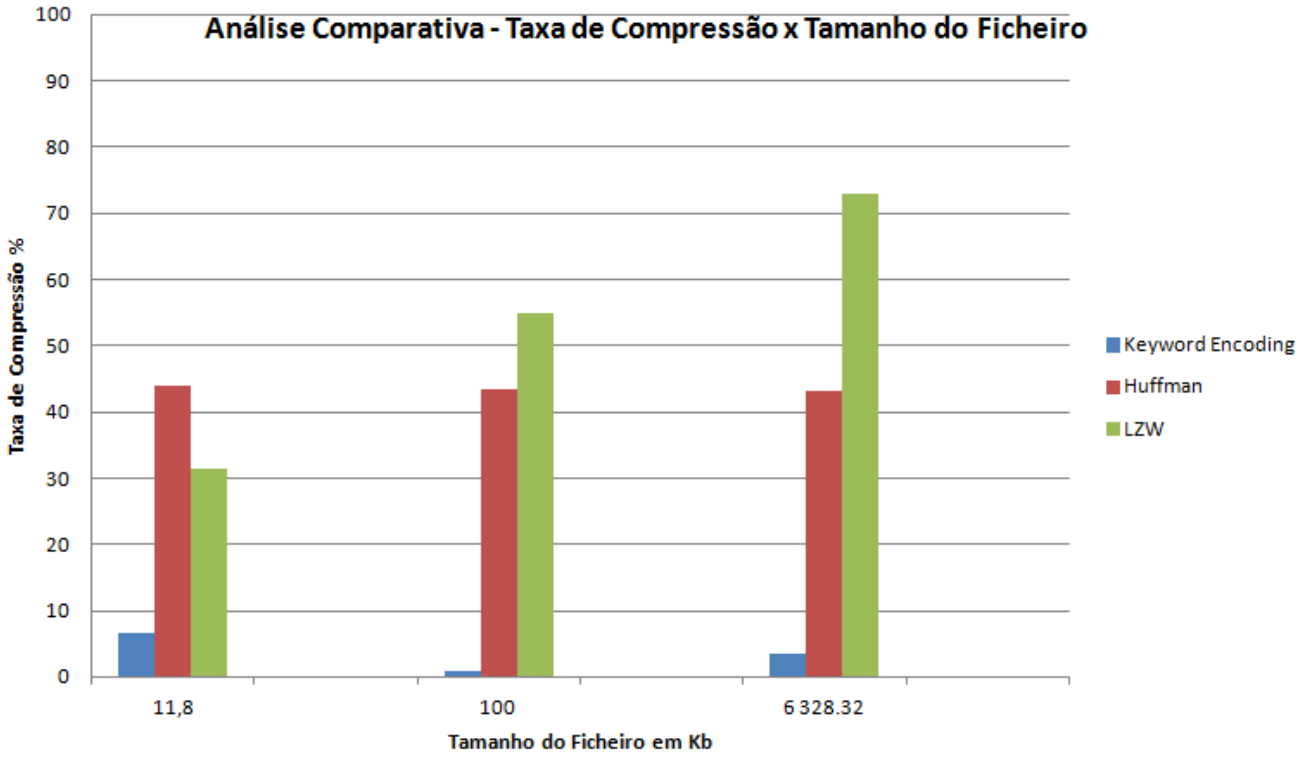

Figura 2 – Gráfico Comparativo Taxa de Compressão x Tamanho Ficheiro

Com base na análise comparativa que se pode efectuar com base nos dados recolhidos, percebe-se, inicialmente, que o método *Keyword Encoding* apresenta uma taxa de compressão muito pequena, quase nula. Já os métodos de *Huffman* e *LZW*, mostraram-se bastante eficientes. Para arquivos de maior tamanho, é de realçar a maior capacidade de compressão do método *LZW*, porém o processo de compressão revela-se bastante mais lento que o método de *Huffman*, que apresenta taxas de compressão por volta dos 50%.

3. Conclusões

Com base no trabalho desenvolvido, os elementos do grupo consideram que conseguiram atingir os objectivos inicialmente propostos.

A aplicação vai ao encontro daquilo que foi pedido e os elementos do grupo conseguiram também aprender os vários algoritmos utilizados.

Quanto à implementação do código, esta decorreu sem grandes problemas, alguns entraves surgiram no decorrer da escrita do código, mas com o tempo e a dedicação do grupo estes foram sempre ultrapassados.

O grupo de trabalho funcionou bem, as tarefas foram distribuídas ficando cada elemento responsável por partes igualmente importantes no projeto final. Quando foi necessário reunimos para discutir ideias e procurar soluções, sendo que tudo decorreu da melhor forma, sem atritos e com todos os elementos do grupo a trabalharem bem e harmoniosamente.

4. Recursos

4.1. Bibliografia

1. Slides da unidade curricular de Concepção e Análise de Algoritmos.
2. "*Introduction to Algorithms*", Second Edition, Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, The MIT Press, 2001.
3. "LZW – Wikipédia, a enciclopédia livre" - http://pt.wikipedia.org/wiki/LZW
4. "O Processamento digital de imagens", Ogê Marques Filho e Hugo Vieira Neto - http://www.scribd.com/doc/43911348/95/Codificacao-LZW-Lempel-Ziv-Welch
5. "Algoritmo LZW (Lempel –Ziv-Welch)" - http://www.decom.fee.unicamp.br/dspcom/EE088/Algoritmo_LZW.pdf

4.2. Software

- Eclipse IDE