

Relatório do Segundo Trabalho de Conceção e Análise de Algoritmos

Francisco Veiga, up201201604@fe.up.pt

João Cabral, up201304395@fe.up.pt

João Mota, up201303462@fe.up.pt

Faculdade de Engenharia da Universidade do Porto

1 de junho de 2015

Conteúdo

1	Introdução	3
2	Problemas a abordar	3
3	Casos de Utilização	3
4	Formalização do problema	4
4.1	Algoritmo de Huffman	4
4.2	Algoritmo LZW	4
5	Considerações sobre <i>Run Length Encoding</i>	5
5.1	Compressão de textos	5
5.2	Compressão de imagens	5
5.3	Codificação	5
5.4	Decodificação	6
6	Descrição dos Algoritmos	6
6.1	Huffman	6
6.2	LZW	7
7	Análise de Complexidade	8
7.1	Huffman Tree	8
7.2	LZW	8
8	Análise Empírica	9
8.1	Taxas de compressão	9
8.2	Tempo de execução	9
9	Considerações sobre a realização do trabalho	11
9.1	Principais dificuldades sentidas	11
9.2	Contribuição dos elementos do grupo	11

Lista de Figuras

- 1 Variação da performance do algoritmo de compressão usando codificação de *Huffman* em função do tamanho da mensagem a codificar. 10
- 2 Variação da performance do algoritmo de compressão usando *LZW* em função do tamanho da mensagem a codificar. 10
- 3 Variação da performance do algoritmo de compressão usando *RLE* em função do tamanho da mensagem a codificar. 11

Lista de Tabelas

- 1 Compressão obtida por cada algoritmo implementado sobre diferentes tipos de ficheiros 9

1 Introdução

No contexto da unidade curricular de Conceção e Análise de Algoritmos, foi solicitada a conceção de um programa de compressão de ficheiros.

2 Problemas a abordar

Para o presente trabalho foi pedida a conceção de um programa de compressão, de seu nome ‘LeZip’, que se baseia em três algoritmos: *Algoritmo de Huffman*, *Algoritmo de Lempel-Ziv-Welch* e *Run Length Encoding*.

3 Casos de Utilização

Está implementada a funcionalidade que permite ao utilizador passar uma pasta ao programa e escolher o algoritmo de compressão desejado. O programa irá gerar uma pasta semelhante com os ficheiros comprimidos. O mesmo programa poderá também executar a operação inversa, gerando uma pasta semelhante à original.

É de realçar que o algoritmo *Run Length Encoding* apenas se encontra funcional para a compressão de texto, obtendo resultados errados para outros tipos de ficheiros.

4 Formalização do problema

4.1 Algoritmo de Huffman

Inputs

- Um alfabeto $A = \{a_1, a_2, \dots, a_n\}$ de caracteres presentes no ficheiro a comprimir.
- Uma lista de frequências $F = \{f_1, f_2, \dots, f_n\}$ onde $f_i = \text{peso}(a_i)$, $1 \leq i \leq n$.

Outputs

- Uma lista de códigos binários $C(A, F) = \{c_1, c_2, \dots, c_n\}$ onde c_i é o código de a_i , $1 \leq i \leq n$ e $|c_i|$ é o comprimento em bits de c_i .

Função Objetivo

Seja $L(C) = \sum_{i=1}^n f_i |c_i|$. Pretende-se encontrar $C(A, F)$ que minimize $L(C)$.

4.2 Algoritmo LZW

Inputs

- Um dicionário de símbolos $D = \{d_1, d_2, \dots, d_n\}$
- Uma mensagem $M = \{m_1, m_2, \dots, m_l\}$ tal que $\forall m \in M (\exists d \in D (m = d))$

Outputs

- O dicionário D acrescido de algumas combinações de símbolos consecutivos de M .

5 Considerações sobre *Run Length Encoding*

Não é feita uma análise formal do algoritmo de Run Length Encoding. Sobre o mesmo é portanto elaborada a seguinte explicação.

O RLE é uma técnica de compressão, que permite comprimir cadeias de caracteres onde existam sequências longas de caracteres repetidos.

O princípio deste algoritmo é simples, quando temos a ocorrência de uma repetição continuada de um carácter, por exemplo, BBBBBBBB é possível representá-lo da seguinte forma 7B. No entanto não podemos simplesmente substituir no meio de um texto a sequência de caracteres por números porque iria ser extremamente difícil detetar situações em que fossem usados algarismos no texto.

Neste caso temos que distinguir se o algarismo já estava presente no texto ou se foi introduzido pela codificação. Assim se usarmos por exemplo um carácter especial podemos identificar o início da codificação por exemplo *7B. Esta técnica só é eficiente se a sequência tiver um tamanho maior de 3, além disso o carácter especial não pode ser um dos caracteres que ocorrem no texto.

5.1 Compressão de textos

Para a compressão de textos este método não é muito eficiente. Por exemplo, na Europa não são muito comuns as repetições de três ou mais letras. Repetições de 4 caracteres iguais só ocorreriam em tabelas, quadros, ou com caracteres especiais tais como os finais de linha, espaços e tabulações.

5.2 Compressão de imagens

Na compressão de imagens esta técnica é mais promissora pois imagens apresentam maiores áreas continuas da mesma cor.

5.3 Codificação

Admita-se a seguinte mensagem a codificar:AAAAAAACVBDDDDDDDDDDDD

Este algoritmo recebe um ficheiro de texto com sequências de caracteres, lê-os um a um, e na eventualidade de haver repetição continuada de um carácter é aplicado o algoritmo, caso contrário o mesmo é escrito.

No exemplo em questão o resultado seria 7ACVB*10D.

5.4 Descodificação

Admita-se a seguinte mensagem a descodificar: *7ACVB*10D

Este algoritmo recebe um ficheiro comprimido, lê carácter a carácter, e caso encontre o carácter especial lê o carácter a seguir, que é o numero de caracteres a escrever do carácter a seguir, se não escreve o carácter.

No exemplo em questão o resultado seria AAAAAAACVBDDDDDDDDDD.

6 Descrição dos Algoritmos

6.1 Huffman

```
procedure HUFFMAN-TREE( $f : [f_1, f_2, \dots, f_n], f_i = w(i)$ )  
   $T \leftarrow$  Árvore Binária vazia  
   $Q \leftarrow$  Fila de prioridade iniciada com os nós da lista  $f$   
  for  $k = 1$  to  $k = n - 1$  do  
     $left \leftarrow$  EXTRACT-MIN( $Q$ )  
     $right \leftarrow$  EXTRACT-MIN( $Q$ )  
     $node \leftarrow$  CREATE-NODE( $T, left, right$ )  
    INSERT-NODE( $T, left, right$ )  
    INSERT-QUEUE( $node$ )  
  end for  
  return  $T$   
end procedure  
procedure HUFFMAN-ENCODE1( $in, out, T : \text{HuffmanTree}$ )  
  for  $character \in in$  do  
     $out \leftarrow$  FIND( $T, character$ )  
  end for  
end procedure
```

¹O procedimento para descodificar é análogo

6.2 LZW

```
procedure LZW-ENCODE(inputstream, outputstream)
   $d \leftarrow$  INITIALIZE-DICTIONARY
   $curr \leftarrow 0$ 
   $next \leftarrow 1$ 
   $s \leftarrow$  inputstream
  while  $curr < s.size$  do
     $currString \leftarrow$  SUBSTRING( $s$ ,  $curr$ ,  $next$ )
    while  $currString \in d$  do
       $next \leftarrow next + 1$ 
       $currString \leftarrow$  SUBSTRING( $s$ ,  $curr$ ,  $next$ )
    end while
     $output \leftarrow$  substring( $s$ ,  $curr$ ,  $next-1$ )
     $outputstream \leftarrow$   $output$ 
    INSERT( $d$ ,  $currString$ )
     $curr \leftarrow next - 1$ 
     $next \leftarrow curr + 1$ 
  end while
end procedure

procedure LZW-DECODE(instream, outstream)
   $d \leftarrow$  INITIALIZE-DICTIONARY
   $bits \leftarrow$  outstream
   $s \leftarrow$ 
   $temp \leftarrow$ 
  while  $|bits| \neq 0$  do
     $NumberOfBits \leftarrow$  CEILING( $\log_2(|d|)$ )
     $code \leftarrow$  NEXT-CODE( $bits$ )
    if  $code \in d$  then
       $temp \leftarrow s$ 
       $s \leftarrow$  FIND( $d$ ,  $code$ )
       $newEntry \leftarrow temp$ 
    else
       $newEntry \leftarrow s$ 
    end if
     $newEntry \leftarrow$  CONCATENATE( $newEntry$ , SUBSTRING( $s$ , 0, 1))
    INSERT( $d$ ,  $newEntry$ )
     $outstream \leftarrow s$ 
  end while
end procedure
```


7 Análise de Complexidade

7.1 Huffman Tree

Seja n o número de símbolos de uma mensagem sobre um alfabeto de dimensão m . É necessário percorrer duas vezes a mensagem, uma para contar as frequências de cada símbolo e outra para codificar a mensagem. Para construir a árvore de Huffman propriamente dita é necessário inserir cada símbolo do alfabeto na fila de prioridade, o que é uma operação realizável em tempo logarítmico. Temos portanto que a construção da árvore tem uma complexidade $O(m \log(m))$. Acresce a necessidade de percorrer toda a mensagem a codificar e procurar o seu código na árvore binária, o que no caso médio é uma operação $O(\log(m))^2$. Repetindo a operação para todos os caracteres da mensagem $O(n \log(m))$. Obtém-se portanto uma complexidade $O((m + n) \log(m))$. O valor de m é limitado superiormente pelos 8 bits usados na representação de símbolos, pelo que a sua inclusão na análise assintótica é de interesse questionável. Na prática o tempo de execução é superiormente limitado pelo tamanho da mensagem, para qualquer mensagem cuja dimensão justifique o overhead associado ao uso de uma Huffman Tree.

7.2 LZW

Seja n o número de símbolos da mensagem. O algoritmo percorre a mensagem a codificar uma vez. Para cada símbolo o algoritmo efetua uma inserção num dicionário implementado como uma *Árvore Vermelho-Preto*. Esta operação tem uma complexidade logarítmica pelo que o Algoritmo tem complexidade de codificação $O(n \log(n))$. Para decodificar uma mensagem o processo é análogo, sendo necessário fazer uma procura no dicionário, cuja complexidade é também logarítmica. Obtém-se portanto um tempo semelhante $O(n \log(n))$.

²As árvores de Huffman são por natureza árvores desequilibradas. No pior caso a complexidade da operação pode portanto aproximar-se de $O(n)$, para uma árvore degenerada cujo comportamento se assemelha na prática ao de uma lista ligada.

8 Análise Empírica

8.1 Taxas de compressão

Taxas de compressão dos algoritmos implementados					
	Tipo de ficheiro utilizado				
	PDF	Imagem	Executável	Texto	Som
Huffman	87.76%	55.30%	117.18%	72.30%	118.08%
LZW	48.32%	36.65%	116.71%	43.37%	117.67%

Tabela 1: Compressão obtida por cada algoritmo implementado sobre diferentes tipos de ficheiros

labelmy-label

Por apenas se encontrar funcional para a compressão de ficheiros de texto, onde apresenta uma taxa de compressão quase irrelevante, o algoritmo *Run Length Encoding* não foi incluído nesta análise.

8.2 Tempo de execução

Para efetuar uma análise empírica do tempo de execução de cada programa foram gerados ficheiros aleatórios com tamanhos progressivamente maiores e medidos os respetivos tempos de execução. Os resultados obtidos foram representados sob a forma dos gráficos das figuras 1, 2 e 3.

Os resultados da análise empírica parecem confirmar os resultados obtidos na análise teórica efetuada na secção 7 da página 8, aumentando de forma *quasilinear*, no caso do *LZW* e *Huffman* e linear no caso do *RLE* com o tamanho da mensagem a codificar.

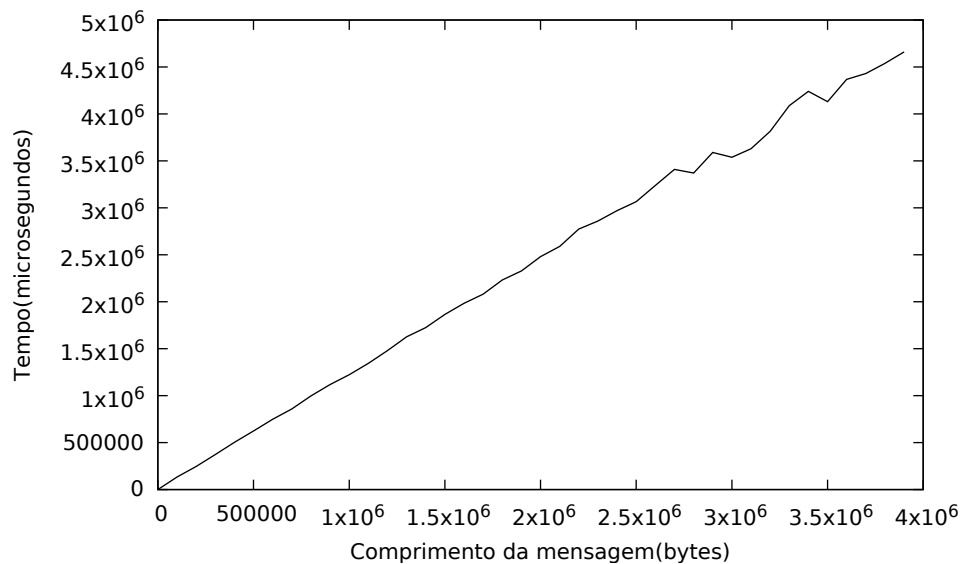


Figura 1: Variação da performance do algoritmo de compressão usando co-dificação de *Huffman* em função do tamanho da mensagem a codificar.

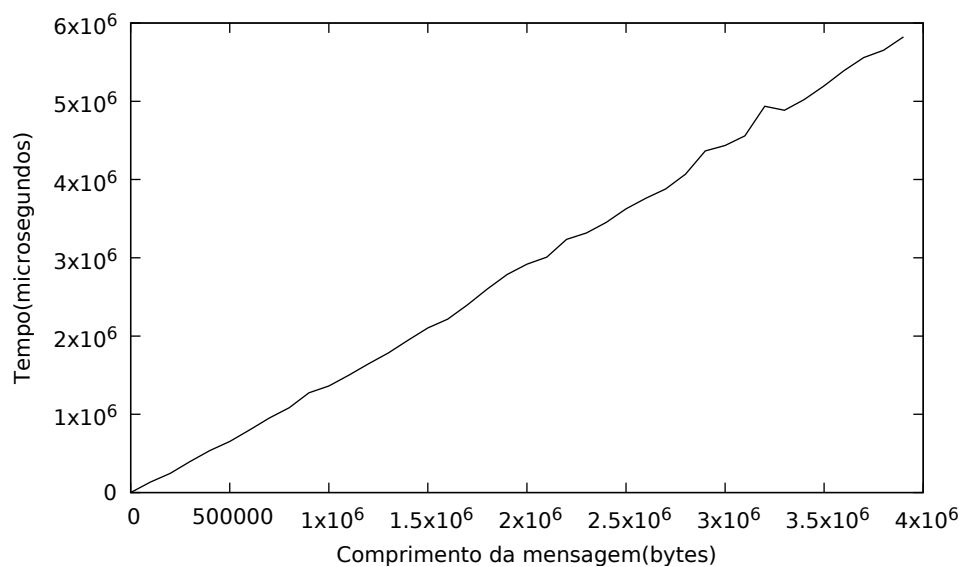


Figura 2: Variação da performance do algoritmo de compressão usando *LZW* em função do tamanho da mensagem a codificar.

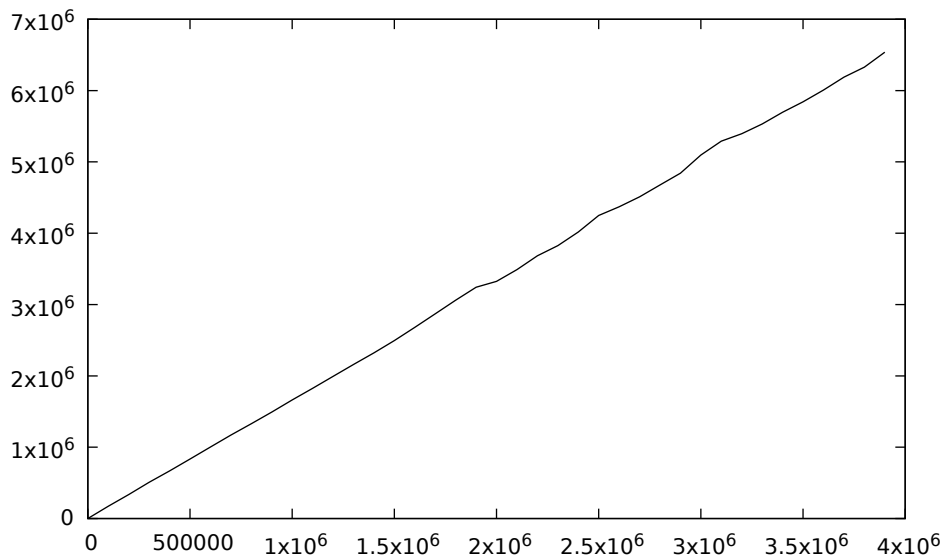


Figura 3: Variação da performance do algoritmo de compressão usando *RLE* em função do tamanho da mensagem a codificar.

9 Considerações sobre a realização do trabalho

9.1 Principais dificuldades sentidas

A principal dificuldade sentida na realização prendeu-se com um problema no algoritmo *Run Length Encoding*

9.2 Contribuição dos elementos do grupo

Todos os membros contribuíram para o trabalho, desenvolvendo funcionalidades do programa, bem como colaborando na formalização matemática do problema, na implementação de algoritmos, na descrição da solução e na análise empírica dos tempos de execução.

Francisco Veiga Implementou o algoritmo *Run Length Encoding* e escreveu a sua descrição para o relatório.;

João Cabral Implementou o algoritmo de *Huffman*, colaborou na implementação do *LZW* e elaborou o presente relatório.

João Mota Implementou o algoritmo *LZW*, colaborou na implementação do *Huffman* e escreveu o pseudocódigo presente neste relatório a respeito do mesmo. Implementou a interface do utilizador com o programa.