Introdução à codificação multimédia (compressão)

→ A compressão é absolutamente essencial para permitir o uso dos tipos de media não estruturados: AUD, VID, IMG (Revisão de exemplos de motivação)

→ Objetivos da codificação multimédia:

a) Para reduzir o espaço de armazenamento ocupado (mede-se em bytes)
b) Para transmitir/ stream em redes computadores reduzindo largura de banda (mede-se em bps)

Motivação:

Exemplo 1
Calcular a quantidade de dados armazenados numa frame de vídeo PAL digitalizada (uma imagem bitmap)

Carateristicas PAL:
25fps; 576x768 ; 24bpp(RGB)

Cada componente R,G e B possui 255 tonalidades (1 byte)

00000000 – mais escuro
.
.
.
11111111 - mais claro

Espaço = 576*768*24=10616832 bits/8 bytes/1024 kbyte = 1296 kbytes

Exemplo 2
Qual é a largura de banda necessária para transmitir este vídeo?

LB=1296kb(1 frame)*25fps = 32400 kbytes/s = (32400*8)/1024=253,125 Mbit/s ← bit rate ou débito binário (muito grande para a qualidade)

Exemplo 3
Considerando o espaço ocupado num suporte ótico DVD-R, quantos minutos deste vídeo seria capaz de armazenar

tempo= capacidade do disco ótico/débito binário do vídeo = (4,377Gbytes)/(32400kbytes/s) = (4,377*1024*1024 kbytes)/ (32400 kbytes/s)=141.65s/60=2,36 min

Exemplo 4
Considerando que temos uma longa metragem(90 min) deste vídeo. Que espaço ocupa no drive?

Espaço=32400kbytes/s*90*60=(174960000 kbytes)/(1024*1024)=166.85 Gbytes

Modos e categorias para compressão

a) Quanto ao modo como a informação é comprimida, isto é, quanto à forma como os algoritmos comprimem a informação:

C.S.P (compressão sem perdas - lossless) – técnicas que apenas otimizam a representação binária, removendo a redundância

As técnicas de C.S.P permitem recuperar a informação original após compressão e subsequente descompressão.

Fluxo de dados de saída = Fluxo de dados de entrada

Usada para compressão de:
TXT
Gráficos vetoriais
Animação
Imagem c/ qualidade
Código executável

C.C.P (compressão com perdas – lossy) – técnicas que eliminam informação imperceptível ao ser humano

Neste caso, a compressão é irreversível, conduzindo à perda de alguma informação de alguns dados originais.

Fluxo de dados de saída <> Fluxo de dados de entrada

Usada para compressão de:
Fotografias genéricas
Audio p streaming
Vídeo p streaming

Esquema geral para os codecs multimédia

AUD VID IMG	Fluxo de dados original (s/ compressão)	→ **Codificador** →	**Armazenamento ou transmissão** (discos, streaming…)	→ **Descodificador** →	Fluxo de dados de Saída

Fluxo de dados comprimidos = Sequência de códigos binários

Rácio de compressão = nº de bytes de entrada (original) / nº de bytes do fluxo comprimido
(Rácio = Bytes IN / Bytes coded)

Quanto maior for o rácio de compressão, mais comprimido fica o fluxo de dados original, isto é, menor fica o espaço ocupado pelo fluxo de dados comprimidos (codificado)

b) As categorias de compressão permitem classificar os algoritmos/técnicas de compressão/ codecs quanto à forma como o algoritmo encara a informação a comprimir

1º Técnicas de codificação de entropia (Entropy encoding)
→ Não têm em consideração a natureza dos dados a comprimir
→ Tratam o fluxo de dados de entrada como uma sequência de símbolos genéricos (1 símbolo = 1 byte)
→ A informação a comprimir é uma sequência binária: ignoram a semântica dos dados!
2º Técnicas de codificação de fonte (Source encoding)
→ Têm em consideração o tipo de media que estão a comprimir
→ As transformações que estas técnicas operam dependem do tipo de dados que comprimem (AUD, VID, IMG)

Categoria	Característica	Modo de compressão	Operação	Exemplos
Entropy encoding	Ignoram a semântica dos dados a comprimir	C.S.P	Remoção de redundância ou otimização da representação binária	ZIP, RAR, LZ77, LZW
Source encoding	Atendem ao tipo de media a comprimir	C.C.P ou C.S.P	QUANTIFICAÇÃO - Sequências de amostras sonoras - Pixeis de imagem - Sequências de frames de vídeo	JPEG, MPEG

Não confundir o formato contentor com a técnica de compressão:
.avi → Vídeo comprimido em mpeg-2/ mpeg-4 / H.264 /H.265
.zip → deflate / huffman/ LZW / LZ77

Exemplo de um codificador de entropia simples:

Pretende-se comprimir o seguinte saldo bancário:

43000000000091 => 43**!10**91 (a flag ! Indica o nº de zeros)

Rácio = 14 bytes/6bytes (a contagem “10” só ocupa um byte)

Entropy encoders	Types: a) Técnicas de supressão de sequências repetitivas (RLE) b) Técnicas de codificação estatística (comprimento variável) c) Técnicas baseadas em dicionários (comprimento fixo)

Source encoders	a) Técnicas baseadas em transformadas b) Técnicas de codificação diferencial c) Técnicas de quantificação vetorial

1.3 Técnicas de codificação de entropia (claude shannon, teoria da informação, 1956)

a) Supressão de sequências repetitivas (+simples, + básicas, + antigas)
→ Produzem códigos (comprimidos) de comprimento fixo
→ Princípio de funcionamento
a) Parse and detect
→ Detetar sequências repetitivas de bits ou bytes
b) Subsequente substituição por um valor que representa o número de ocorrências

→ Há duas formas mais comuns de implementar
1º) Substituição de zeros ou espaços
→ Exige que se especifique o tipo de dados de entrada (numéricos ou texto)
→ Apenas funciona num byte pré-determinado 0 ou |_| (espaço)
→ Uma sequência repetitiva de zeros/espaços substitui-se por um TOKEN: Flag seguida do nº de ocorrências
Ex: 7420000000000005 => 16 bytes

Seq. comprimida: 742!1205 => 5bytes

O token (!12) só ocupa um byte

Rácio= 16/5=2,66:1 (Lê-se 2,66 para 1)

Problemas:
-Ineficiente (só funciona para um token)
-Quanto existem repetições superiores a 255 requer novo token

2º) Método RLE: Run-lenght Encoding
→ Permite substituir qualquer char ou símbolo repetitivo modificando o token:
! <n> <c>
! - flag
<n> - nº repetições
<c> - char que se repete

=> Substitui qualquer sequência repetitiva por 1 token de 3 bytes
=> Só tem interesse quando o nº repetições >= 4

Exemplo:
ABCCCCCCABCABC Rácio=14/11=1,28:1

RLE: AB!6CABCABC

Os métodos baseados em dicionários permitem substituir sequências de vários chars idênticos.

A técnica RLE é muito usada na compressão de imagens (para guardar originais de forma +eficiente: BMP, TIFF, PSD)

=> Exemplo para uma linha de Imagem 8bpp

75|75|75|75|75|75|10|45|58|30|30|30|30|8|8|8|8|8|8|8|3

Há 2 implementações possíveis para imagens:
6,75,10,45,58,4,30,5,8,3 Contagens

Como distinguir uma contagem de um valor de contagem do token?

a) Se a imagem apenas contiver 128 tonalidades distintas usa-se o bit + significativo para assinalar a flag (1)

000000000
cores originais
011111111

10000000
cores repetidas (contagem)
11111111

b) Se houver mais do que 128 tonalidades, usa-se a tonalidade menos frequente (255) como flag

Pode ser necessário reduzir o número de tonalidades de 256 para 255, libertando o ultimo byte p/FLAG

Ex: 6,75,10,45,58,4,30,5,8,3

255,6,75,10,45,58,255,4,30,255,5,8,3

b) Codificação estatística (comprimento variável)

Símbolo de comprimento fixo

Codificador estatístico (Uso de modelo de probabilidades p/ os simbolos)

Código binário comprimido de comprimento variável

1º) Identificar os símbolos mais e menos frequentes (padrões de bits para símbolos mais extensos que um byte)

2º) Codifica os símbolos de acordo com o seguinte princípio:
2.1) Os símbolos mais frequentes são codificados com menos bits
2.2) Os símbolos menos frequentes são codificados com mais bits

Requisito: exigem a construção de 1 “Codebook” que faz corresponder os símbolos aos respetivos códigos

Respeitando a entropia dos dados originais

As implementações mais comuns:
a) Substituição de padrões
“Multimédia” → *M (substituição dos padrões que aparecem mais vezes)

b) O codificador de comprimento variável
Basear a obtenção de códigos em àrvores binárias
Ex: Shannon e Huffman

c) Codificação aritmética
Trata-se o ficheiro todo como um único símbolo, portanto gera-se sempre um único código comprimido

Exemplo: Comparar a eficiência de codificação de um codificador estatístico (Huffman) com um código otimizado de comprimento fixo.
Construi-se 1 ficheiro de texto com 100 000 chars gerados aleatoriamente a partir do alfabeto {a,b,c,d,e,f}

a) Usando ASCII → 8 bits por char (8bpc) → 800 000 bits

b) Usei 1 código otimizado de comprimento fixo → 3 bits por char → 300 000 bits

c) Usei 1 codificador estatístico de Huffman → 1,3 e 4 bpc → 224 000 bits

Freq.	45000	13000	12000	16000	4000	5000	100 000
	A	B	C	D	E	F	Total
ASCII	1B	1B	1B	1B	1B	1B	800 000 bits
Código otimizado	000	001	010	011	100	101	300 000 bits
Código de Huffman	0,45 0	0,13 101	0,12 100	0,16 111	0,04 1101	0,05 1100	224 000 bits

224000/100000=2,24 bpc

d) Codificação baseada em dicionários

Conjuntos de símbolos descomprimidos de comprimento variável

↓

Codificador baseado em dicionários

↓

Código binário comprimido de comprimento fixo

→Constrói-se um dicionário com códigos de comprimento fixo onde se agrupam sequências repetitivas de símbolos de entrada
→ Exemplos: LZW, LZZZ, LZZ8, LZ55
→ Normalmente o dicionário de base (de partida) é o código ASCII e, a seguir, acrescentam-se grupos de símbolos aos códigos seguintes,
Exemplo: Dicionário com 12 bpc (12 bits por char)
Possui 4096 entradas das quais as 256 primeiras são ASCII

A – 12 bits
AB – 12 bits
ABCABC – 12 bits

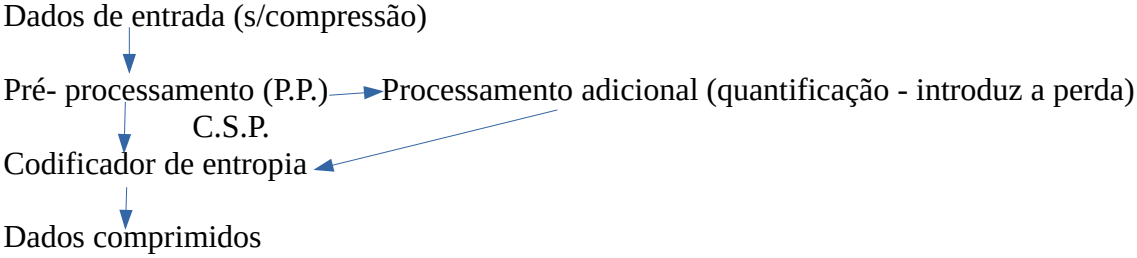
Compressão sem perdas

→ Métodos de codificação de entropia (métodos baseados na noção de entropia introduzida por C. Shannon na Teoria da Informação.)

Os métodos a ser estudados (métodos = algoritmos = codecs = técnicas):
a) Algoritmo de Shannon-Fano
b) Algoritmo de Huffman
c) Codificação Aritmética
d) Método LZW

Objetivo:
Eliminar a redundância da informação e otimizar a respetiva representação binária.

Modelo geral para os codecs multimédia:



Pela teoria da informação de Shannon definimos uma fonte de informação que gera n símbolos distintos aleatórios (r1,r2,r3,…,rn).

Para cada símbolo existe uma probabilidade de ocorrência :

Si → Pi (Si = símbolo de índice i, Pi = probabilidade de ocorrência)

Neste caso, Shannon definiu a auto-informação do simbolo (Si)

$I(Si) = \log_2(1/Pi) = -\log_2(Pi)$

Sabendo que, neste caso, a informação mede-se em bps (bits por símbolo)

Neste contexto, a Entropia da fonte S calcula-se do seguinte modo:

H(S) = Σ (i entre 1 e n) Pi* I(si) (bps)

Pela teoria da informação, Shannon demonstrou que, se os n símbolos forem todos distintos, então o nº médio mínimo de bits por símbolo que é necessário para codificar a informação original sem perdas é dado pela ENTROPIA de S

H(S) <= Nº médio mínimo de bits por símbolos

Ex:
A – 0
B – 110 Tamanho médio do código = Σ (i entre 1 e n) li*Pi
C – 111

Exemplo: Pretende-se codificar a sequência de símbolos:

4 5 8 6 4 7 8 9 4 8

Para determinar a entropia desta sequência é necessário saber a probabilidade de ocorrência de cada símbolo.
Quantos símbolos distintos existem?

S= {4,5,6,7,8,9} → 6 símbolos distintos

1ª Hipótese: Supondo que todos ocorrem com a mesma probabilidade

P1=P2=P3=P4=P5=P6= 1/6 (as probabilidades são todas iguais)

Neste caso, a entropia de S é dada por:

H (S) = Σ (1 até 6) Pi * log2(1/Pi) = 6*((1/6)*log2(6)) = 2,585 bps

2ª Hipótese: Supondo que a probabilidade de cada símbolo pode ser estimada através da respetiva frequência relativa de ocorrência:

P4 = P8 = 3/10
P5 = P6 = P7 = P9 = 1/10

Neste caso, a entropia de S é dada por:

H (S) = Σ (1 e 6) Pi*log2(1/Pi) = 2* (3/10*log2(10/3)) + 4*(1/10*log2(10))= 2,371 bps

Ao comparar o resultado das duas hipóteses concluímos que:

Quanto menos homogéneo for o modelo de probabilidades mais baixo é o valor da entropia e menos bits em média por símbolo são necessários p/codificar sem perdas.

Exemplo: Pretende-se comprimir sem perdas, com um codificador de entropia, duas imagens A e B de cor indexada (8bpp)
As imagens caracterizam-se pela respetiva distribuição de frequências (histograma de cores):

Imagem A

O histograma representa a igual frequência dos 256 índices da corresponder (Distribuição homogénea ou uniforme de probabilidades)

Imagem B

O histograma representa que ⅓ dos pixéis tem a côr 30 e os restantes ⅔ têm a cor 200

Entropia da imagem A:

H(S) = Σ (1 e 256) Pi*log2(1/Pi) = 256*1/256*log2(256) =8 bps → Não há compressão sem perdas possível

Rácio = 8bpp à entrada/8 bps à saida = 1:1

Entropia da imagem B:

H(S) = Σ (1 e 2) Pi*log2(1/Pi) = ⅓ *log2(3) + ⅔ * log2(3/2) = 0,918 bps

Rácio = 8bpp à entrada/ 0,918 bps à saida = 8,69:1

Revisitando a Imagem A para aplicar a compressão → Verifica-se que possui 256 pixéis distribuídos do seguinte modo:

0,1,2,3….,255

Imagem A

↓

Pré processador (vai diminuir a entropia à imagem A) – Aplica-se o codificador diferencial (este substitui o valor real de cada amostra pelo seu valor predito e calcula a diferença entre ambos

↓

Codificador de entropia

Uma vez que a imagem tem as cores seguidas na sua representação (1,2,3,…,255)

Fica: 0, (1-0), (2-1)….

H(Img a pré-proc. C.D.) =(1/256 *log2(256)) + (255/256 * log2(256/255)) = 0,0369 bps

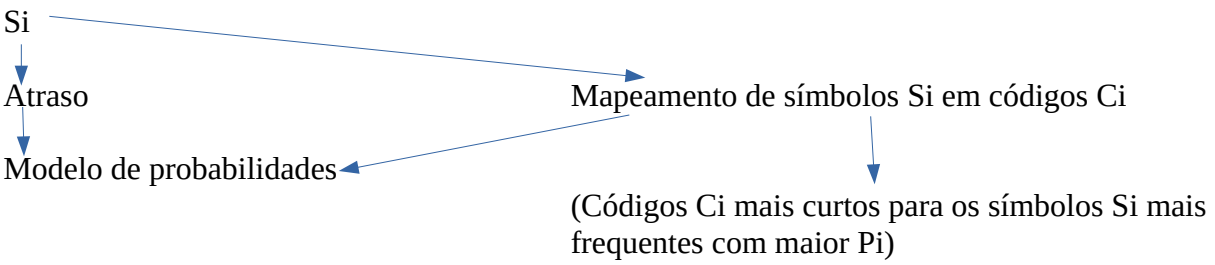
Rácio = 8/0,0369 =222,22:1

Tipos de pré- processamento:

Codificação diferencial

Transformados p/ frequências
Run LenghEncoding (RLE)

Modelo geral dos codificadores de entropia de comprimento variável:



<u>Codificador de Entropia (Algoritmos)</u>
Revisão:
a) Codificadores de comprimento variável (Cod. Estatistica – gasta mais bits no que ocorre menos): Shannon-Fano/ Huffman
b) Codificadores de comprimento fixo (Cod. baseada em dicionários): LZW (família LZ)
c) Codificação aritmética (Cria um código único): Aproxima-se mais da **<u>entropia</u>**

Nº médio mínimo de bits por símbolo que se pode gastar para representar a informação binária sem perdas (H(S))

1º Algoritmo de Shannon-Fano
→ Produz códigos de comprimento variável
→ Abordagem TOP-DOWN
→ Baseia-se na construção de uma árvore binárias

Etapas do algoritmo:

1º Passo – Ordenas todos os símbolos si do alfabeto por ordem dos respetivos pi (ou fr)

2º Passo – Subdividir, recursivamente, os simbolos si em 2 subgrupos, de modo que a soma das pi (ou fr) dos si em cada subgrupo seja equivalente. Repetir até que cada subgrupo contenha apenas um símbolo si (folha)

Exemplo:

Pretende-se codificar a string “VIVER” com SF

Alfabeto: {V,I,E,R}

Si	Fr (frequência relativa)	CódigoSF
V	2/5 = 0,4	00
I	1/5= 0,2	01
E	0,2	10
R	0,2	11

V,I,E,R			
V,I (0,6) - 0		E,R (0,4) - 1	
V(0,4) - 0	I(0,2) - 1	E(0,2) - 0	R(0,2) - 1

Para atribuir código binário atribui-se a convenção esquerda do ramo =0, direita =1, depois o código fica o caminha de cima para baixo

Tamanho médio do código de shannon-fano = (somatório para todos os elementos do alfabeto) li*pi = 1*0,4 + 2*0,2+2*(3*0,2))=2bps

Problemas:
→ Incerteza sobre a escolha dos grupos em cada nível
→ Produz várias codificações diferentes com números de bits p/ símbolo diferentes

2º Codificação de Huffman
→ Produz códigos de compressão variável
→ Abordagem Bottom-Up
→ Baseia-se na construção de uma árvore binária
→ Huffman atinge sempre a melhor codificação possível de Shannon-Fano

Etapas do Algoritmo:
1º Passo – Ordenar os símbolos si do alfabeto por ordem decrescente de probabilidade de ocorrência (ou fr)

2º Passo – Unir os dois símbolos de menor probabilidade pi criando um novo símbolo hipotético cujo a probabilidade é a soma de ambas que lhe deram origem. Repetir, recursivamente, até que todos os símbolos façam parte de 1 símbolo hipotético único (raiz)

Exemplo:
Codificar a string “MUSICA!” (!=simbolo de end of file)

Alfabeto{A,C,I,M,S,U,!}

Si	Pi(modelo de probabilidade pré-existente)	Código Huffman
A	0,2	01
C	0,3	11
I	0,05	10101
M	0,05	10100
S	0,1	100
U	0,2	00
!	0,1	1011

1º Passo

Si	Pi(modelo de probabilidade pré-existente)
C	0,3
U	0,2
A	0,2
S	0,1
!	0,1
M	0,05
I	0,05

2º Passo

Si	Pi(modelo de probabilidade pré-existente)
C	0,3
U	0,2
A	0,2
S	0,1
!	0,1
(M,I)	0,1

Repetir o 2º passo

Si	Pi(modelo de probabilidade pré-existente)
C	0,3
U	0,2
A	0,2
(M,I),!)	0,2
S	0,1

Repetir o 2º passo

Si	Pi(modelo de probabilidade pré-existente)
C	0,3
(((M,I),!),S)	0,3
U	0,2
A	0,2

Repetir o 2º passo

Si	Pi(modelo de probabilidade pré-existente)
(U,A)	0,4
C	0,3
(((M,I),!),S)	0,3

Repetir o 2º passo

Si	Pi(modelo de probabilidade pré-existente)
((((M,I),!),S),C)	0,6
(U,A)	0,4

Repetir o 2º passo

Si	Pi(modelo de probabilidade pré-existente)
{Alfabeto}	1

<table>
<tr><td colspan="6">{Alfabeto} (1)</td></tr>
<tr><td colspan="2">0(U,A) – (0,4)</td><td colspan="4">1((((M,I),!),S),C) – (0,6)</td></tr>
<tr><td>0U (0,2)</td><td>1A (0.2)</td><td colspan="3">0(((M,I),!),S) – (0,5)</td><td>1C (0,3)</td></tr>
<tr><td></td><td></td><td>0S (0,1)</td><td colspan="2">1((M,I),!) - (0,2)</td><td></td></tr>
<tr><td colspan="2"></td><td></td><td>0(M,I) – (0,1)</td><td>1! (0,1)</td><td></td></tr>
<tr><td colspan="2"></td><td></td><td>0M (0,05)</td><td>1I (0,05)</td><td></td></tr>
</table>

Passando o código para binário :

1 0 1 0 0 0 0 1 0 0 1 0 1 0 1 1 1 0 1 1 0 1 1

Para descodificar o descodificador vai juntando os bits até encontrar um elemento do alfabeto

<u>Cálculos:</u>

Entropia do alfabeto:

H(S) = (Somatório para todos os elementos do alfabeto) pi*log2(1/pi)
= (0,3 * log2(1/0,3)+2*(0,2*log2(1/0,2)) + 2*(0,1*log2(1/0,1))+2*(0,05*log2(1/0,05))= 2,546 bps

Comprimento médio do código de Huffman=
(Somatório para todos os elementos do alfabeto) li*pi = (2*0,3)+2*(2*0,2) +
(3*0,1)+(4*0,1)+2*(5*0,05) = 2,6 bps

Conclusão: Entropia < Comprimento médio do código

Rácio de compressão:

Rácio = 3 bpc/2,6 bps= 1,15:1

Problema:

Codificar o alfabeto {a,b,c,d,e,f}

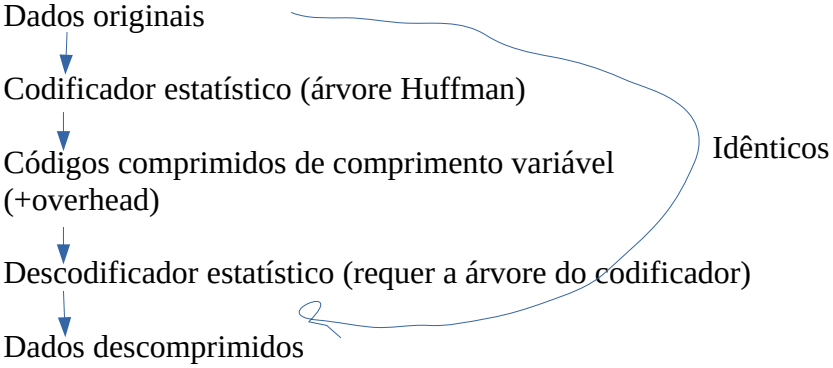
Sendo que:

Si	fr	Código Huffman
a	0,45	0
b	0,13	110
c	0,12	111
d	0,16	101
e	0,09	1001
f	0,05	1000

Comprimento médio do código de Huffman=
(Somatório para todos os elementos do alfabeto) li*pi =
(1*0,45)+(3*0,13)+(3*0,12)+(3*0,16)+(4*0,09)+(4*0,05)= 2,24 bps

---

<u>Descodificação estatística (de Comprimento Variável)</u>
→ 2 algoritmos p/ descodificação de Huffman



1º Algoritmo: Descodificador de bits em série (serial bit decoding) → funciona com base na condição de prefixo
1º passo: Ler o fluxo de dados comprimidos bit-a-bit para um buffer e ir percorrendo a àrvore do codificador até encontrar 1 folha (Si)

2º passo: À medida que cada bit é lido, vai sendo descartado do buffer e ao chegar à folha escreve o símbolo respetivo (Si) no ficheiro descomprimidos
3º passo: repetir 1 e 2 até EOF

Exemplo:

Si	Código
a	0
b	101
c	100
d	111
e	1101
f	1100

A string a codificar é: “aabacadaadfe”
O fluxo comprimido fica: 0 0 101 0 100 0 111 0 0 111 1100 1101
a a b a c a d a a d f e

A taxa de descodificação de símbolos é variável (symbol decoding rate)

Problema em situações em que a informação é visualizada/apresentada em tempo real

2º Algoritmo: Descodificação baseada numa Tabela de LookUp (LUT)

Tem duas fases:

1ª fase: Construção da LUT
2ª fase: uso da LUT para descodificar/ descomprimir o fluxo codificado

Seguindo o exemplo acima:

1ª fase: Construção da LUT
a) Define-se L=n (comprimento em bits do maior código)
L=4 bits
b) Controi-se uma tabela com 2^L entradas e associa-se a cada entrada um tuplo
2^4=16

Endereços	Tuplo (Si, li) → Si= símbolo, li= comprimento do respetivo código
0000	(a,1)
0001	(a,1)
0010	(a,1)
0011	(a,1)

0100	(a,1)
0101	(a,1)
0110	(a,1)
0111	(a,1)
1000	(c,3)
1001	(c,3)
1010	(b,3)
1011	(b,3)
1100	(f,4)
1101	(e,4)
1110	(d,3)
1111	(d,3)

2ª fase: uso da LUT para descodificar/ descomprimir o fluxo codificado
(Lê e descarta o nº de bits indicado)

a) Ler L bits para um buffer
b) Endereçar a LUT com o conteúdo do buffer e descodificar o tuplo (si,li)
c) Escrever o símbolo si no fluxo descomprimido
d) Descartar os primeiros li bits do buffer e ler os li bits em seguintes do fluxo comprimido, e o buffer volta a ficar com L bits

PROBLEMA COMPLETO DE CODIFICAÇÃO/DESCODIFICAÇÃO DE HUFFMAN

A string a codificar tem 39 chars:

Si	Fr	Fa
F	15/39	15
C	7/39	7
P	6/39	6
*	6/39	6
I	5/39	5

Aplicando Huffman

Si	Fa
F	15
(*,I)	11

P	6
C	7

Si	Fa
F	15
(C,P)	13
(*,I)	11

Si	Fa
F	15
((C,P),(*,I))	24

(((F,((C,P),(*,I)))				
0F(15)	1((C,P),(*,I)) - 24			
	0(C,P) - 13		1(*,I) - 11	
	0C(7)	1P(6)	0*(6)	1I(5)

Calculo da entropia do alfabeto = Σ (para todo o alfabeto) pi*log2(1/pi) =

=(15/39*log2(39/15))+(7/39*log2(39/7))+2*(6/39*log2(39/6))+(5/39*log2(39/5)) = 2,186 bps

Comprimento médio do código = Σ (para todo o alfabeto) li*pi =
= (1*(15/39))+(3*((7/39))+2*(3*(6/39))+(3*(5/39))=2,23 bps

Rácio = 3 bits / 2,23 bps = 1,345:1

Agora para descodificar o código

Definir o LUT para o descodificador:
L= 3bits → 2^3 endereços = 8

Endereço	tuplo
000	(f,1)
001	(f,1)
010	(f,1)
011	(f,1)
100	(c,3)
101	(p,3)

110	(*,3)
111	(I,3)

Compressão sem perdas
→ Codificação de comprimento fixo ou baseada em dicionários

Como funciona:

Símbolos de entrada (si) de comprimento variável (agrupando vários símbolos num único)

↓

Codificador baseado em dicionários – Todos os algoritmos da família LZ

↓

Códigos comprimidos de comprimento fixo (ci) – 8 ou 12 bits (nº de bits define a dimensão do dicionário) – 2^n entradas

A compressão é adaptativa porque o dicionário adapta-se ao conteúdo específico de cada ficheiro de entrada que é comprimido

Exemplos de codificadores: formato GIF, compress, deflate, ZIP e RAR

LZW – Lempel-Ziv and Welch

Algoritmo do codificador LZW:

Codificador LZW (fluxo-entrada)

s ← ler próximo símbolo si do fluxo de entradas
Do
    c ← ler próximo símbolo si do fluxo entradas
    if string(s+c) existe no dicionário then
        s ← s+c
    else
        escrever o código s no ficheiro de saída (comprimido)
        adicionar (s+c) ao dicionário com novo código
        s ← c
while(c<>EOF)
escrever o código de s no ficheiro comprimido de saída

Exemplo: Codificar a string "LALAALAILAILAI"

			Dicionário	
s	c	Saída (comprimido)	Código	Símbolo
			1 2 3	A I L

L	A	3	4 (não existe dic)	LA
A	L	1	5	AL
L	A			
LA	A	4	6	LAA
A	L			
AL	A	5	7	ALA
A	I	1	8	AI
I	L	2	9	IL
L	A			
LA	I	4	10	LAI
I	L			
IL	A	9	11	ILA
A	I			
AI	EOF	8		

Cálculo do rácio de compressão resultante
=> Fluxo comprimido 314512498
Neste caso, o dicionário poderia ter o comprimento de 8 bits (porque o dicionário é curto)

Rácio = (14*8)/(9*8)=1,56:1

No caso real, o dicionário contém 12 bits (2^12=4096 entradas)

Rácio= (14*8)/(9*12)=1,04:1

Algoritmo do descodificador LZW: Descodificador LZW (fluxo comprimido)

```
s ← NULL
Do
        c ← ler próximo código do fluxo comprimidos
        cadeia ← símbolo do dicionário correspondente ao código C
        if(cadeia==null) then cadeia=s+s[0]
        escrever no ficheiro de saída (descomprimido) a string
        if(s<>NULL) then
                adicionar ao dicionário (s+cadeia[0]) com novo código
        s← cadeia
while (c<>EOF)
```

Exemplo: Descodificar o código “314512498”

			Dicionário	
s	c	string/saída	código	símbolo
			1 2 3	A I L
NULL	3	L		
L	1	A	4	LA
A	4	**L**A	5	AL
LA	5	**A**L	6	LAA
AL	1	A	7	ALA
A	2	I	8	AI
I	4	**L**A	9	IL
LA	9	IL	10	LAI
IL	8	AI	11	ILA
AI	EOF			

Porém este algoritmo tem um problema!!!!!

Por exemplo com a seguintes string

Codificar : “LAILAALAALAALAA”

s	c	Saída (comprimido)	Código	Símbolo
			1 2 3	A I L
L	A	3	4 (não existe dic)	LA
A	I	1	5	AI
I	L	2	6	IL
L	A			
LA	A	4	7	LAA
A	L	1	8	AL
L	A			
LA	A			
LAA	L	7	9	LAAL
L	A			

LA	A			
LAA				
LAAL	A	9	10	LAALA
A	A	1	11	AA
A	EOF	1		

Descodificar: “312417911”

s	c	cadeia/saida	Código	símbolo
			1 2 3	A I L
NULL	3	L		
L	1	A	4	LA
A	2	I	5	AI
I	4	LA	6	IL
LA	1	A	7	LAA
A	7	LAA	8	AL
LAA	9 PROBLEMA	LAAL	9	LAAL
LAAL	1	A	10	LAALA
A	1	A	11	AA
A	EOF			

O algoritmo inicial não funciona sempre que ocorre no ficheiro de entrada:
L AA L
char+sequência+char(mesmo)+sequência(mesmo)

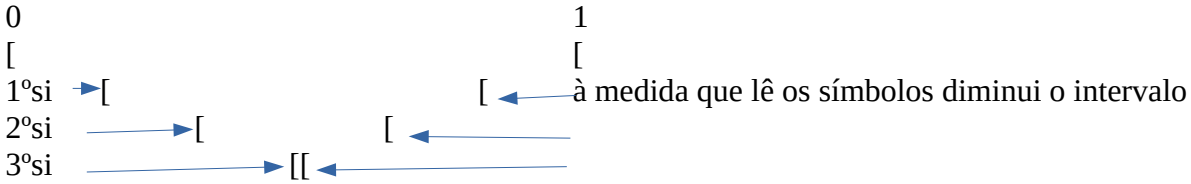
Resumindo:
si (8 bits) – sequências de chars

codificadores LZW (dicionário de arranque – tabela ASCII)

ci(12 bits) 2^12=4096 entradas

<u>Compreensão sem perdas</u>
=> Codificação de entropia: Algoritmo da codificação aritmética (técnica de codificação estatística – comprimento variável – baseada em probabilidades)

Principio de funcionamento: Estabelecer um sub-intervalo com uma precisão tal que define sem ambiguidade o ficheiro de entrada a comprimir



Algoritmo da codificação aritmética (fluxo entrada)

anterior_baixo ← 0
anterior_alto ← 1
largura ← 1 (1-0)
do
ler próximo (si) do fluxo_entrada
anterior_alto ← anterior_baixo + largura***sub-intervalo alto do símbolo**
anterior_baixo ← anterior_baixo +largura***sub-intervalo baixo do símbolo**
largura ← anterior_alto – anterior_baixo
while (si<>EOF)
escrever na saida (fich. Comprimido) um código binário tal que:
anterior_baixo <= conversão para decimal (código aritmético) < anterior alto

Probabilidades comutativas

Exemplo: Alfabeto do exemplo de Huffman (MUSICA!)

Si	Pi	Cód. Huffman
A	0,2	01
C	0,2	11
I	0,05	10101
M	0,05	10100
S	0,1	100
U	0,2	00
!	0,1	1011

Para codificação aritmética é necessário arrancar pelo cálculo das probabilidades cumulativas:

Si	sub-intervalos
M	[0;0,05[
U	[0,05;0,25[
S	[0,25;0,35[
I	[0,35;0,4[
C	[0,4;0,7[
A	[0,7;0,9[
!	[0,9;1[

Problema: Codificar a string “UUSSAC!”

(Ver nos quadros da plataforma)

Resultado final:

[0,0713336;0,0713360[

É necessário escolher um nº que caia dentro deste intervalo:
Esse número tem de ser o que precisa de menos bits para ser representado em binário, e para tal é preciso usar um algoritmo

Algoritmo para encontrar o código binário aritmético:

```
Begin
        k ← 1
        código ← 0 //código binário aritmético
        while(decimal(código)<low)
                atribuir o bit 1 à k-ésima casa fracionaria do código
                se (decimal(código)>high)
                        substituir o k-ésimo bit por 0
                k ← k+1
```

Aplicando o algoritmo para descobrir o nº

K	Código	Decimal
1	0,1	1*2^-1=0,5>high
2	0,01	1*2^-2=0,25>high
3	0,001	2^-3=0,125>high
4	0,0001	2^-4=0,0625<low, mantém o 1
5	0,00011	2^-4+2^-5=0,09375>high
6	0,000101	
...	0,0001001001000011	2^-4+2^-7+2^-10+2^-15+2^-16

Rácio = (7*8)/16= 3,5:1

Comparação com Huffman nesta string:
Codificação com Huffman, seguindo os códigos, fica:

000010010001111011 → 18 bits

Rácio para Huffman = (7*8)/18=3,1

Logo, rácio c. Aritmética >= rácio c. Huffman e comprimento aritmética <= comprimento Huffman