# Pràctica 2: Introducció a la Teoria de l'Informació i la Compressió de Dades

## *Segona part: Codis de Huffman*

### Introducció

Suposem el mateix sistema que l'altre dia, però amb un sistema de codificació entre la font i el destí que compacta (=redueix el tamany de) el missatge, per tal d'augmentar l'eficiència de la comunicació.


**Figura 1- Sistema de comunicació amb compactador**

Tenim l'alfabet de la font

$$s=(s_1,s_2,...,s_N)$$

i el vector de probabilitat associat

$$p=(p_1,p_2,...,p_N)$$

sent $p_i$ la probabilitat de que ocurreixi el símbol $i$.

Sigui ara el vector

$$L=(L(a_1),L(a_2),...,L(a_N))$$

el vector de les longituds en bits dels símbols de l'alfabet.

Definim el *factor de compressió R* com l'esperança de les longituds del símbols:

$$R = \sum p_i \cdot L(a_i) \qquad (1)$$

El problema de codificació de Huffman per aquest model serà el trobar el codi compacte (al qual direm *codi de Huffman*) tal que les longituds dels seus símbols minimitzin (1).

A les següents seccions descriurem l'algorisme de Huffman, veurem una funció MATLAB per calcular la longitud de codi de Huffman i farem alguns exemples, i finalment veurem (i implementarem) el codi de Shannon-Fano, un codi sub-òptim, però d'eficiència semblant al codi de Huffman i molt més fàcil d'implementar.

## L'Algorisme de Huffman

Del vector de probabilitats *p*, suposem que

$$p_1 \geq p_2 \geq \cdots \geq p_N \qquad (2)$$

Ara veurem com l'algorisme de Huffman troba l'arbre de codificació del codi de Huffman per a *p*.

**Pas 1:** Formar el vector de probabilitats d'ordre (N-1)

$$\begin{aligned} \overline{p} &= \left(\overline{p}_1, \overline{p}_2, \cdots, \overline{p}_{N-1}\right) \\ &= \left(p_1, p_2, \cdots, p_{N-2}, p_{N-1} + p_N\right) \end{aligned}$$

Dit d'una altra manera, el vector de probabilitats $\overline{p}$ s'obtén a partir del vector de probabilitats *p*, afegint els dos components més petits de *p* per crear el component N-1 de $\overline{p}$ (la resta de components són els mateixos).

**Pas 2:** Formar l'arbre de codificació del codi de Huffman per la distribució $\overline{p}$ (sí, és un algorisme recursiu). Aquest arbre tindrà N-1 fulles etiquetades amb els components de $\overline{p}$.

**Pas 3:** Trobar una fulla de l'arbre de codificació trobat al pas 2 amb l'etiqueta

$$\overline{p}_{N-1} = p_{N-1} + p_N$$

Fer créixer dues branques des d'aquesta fulla, que acabin en dues fulles noves. Etiquetar una d'aquestes fulles com a "$p_{N-1}$" i l'altra com a "$p_N$". L'arbre resultant té *k* fulles etiquetades amb els components de *p*. Es l'arbre de codificació del codi de Huffman per la distribució *p*.

A l'arbre resultant del pas 3, etiquetem les *k* branques amb $s_1, s_2, \cdots, s_N$ (els símbols del sistema) respectivament, de manera que les etiquetes $s_i$ són consistents amb les etiquetes $p_i$ (o sigui, si una fulla està etiquetada com a $s_i$, llavors l'etiqueta de probabilitat assignada per l'algorisme de Huffman a aquesta etiqueta ha de ser $p_i$).
I a les dues branques que surten de cada node no-terminal assignarem una etiqueta de bit de codi, per exemple amb "0" a la de la dreta i "1" a la de l'esquerra. Es només una convenció, així que podríem perfectament haver escollit fer-ho a l'enrevés.

Un cop fet tot això, el codi de Huffman de l'alfabet *s* codificarà cada símbol $s_i$ seguint el camí arrel-a-fulla que acaba a la fulla etiquetada com a $s_i$, ajuntant les etiquetes de codi de bit trobades a aquest camí per formar la paraula codificada.

Senyalem que si $L_i$ es la longitud de la paraula codificada assignada a $s_i$, per la mateixa mecànica de l'algorisme de Huffman automàticament es complirà que

(precisament per la suposició (2)) . Sí després d'aplicar l'algorisme no és el cas, és que ens hem equivocat al camí.

*Exemple 1*. Apliquem l'algorisme de Huffman al següent vector de probabilitats:

$$p = (0.4, 0.2, 0.2, 0.1, 0.1)$$

Això resulta a les següents reduccions a distribucions de probabilitats més simples:

**Reducció 1:** A $p$ substituirem les dues probabilitats més petites per la seva suma 0.2, formant així la nova distribució de probabilitat

$$p^1 = (0.4, 0.2, 0.2, 0.2)$$

**Reducció 2:** A $p^1$ substituirem els valors 0.2, 0.2 del mig per la seva suma 0.4, de manera que formen la distribució de probabilitat

$$p^2 = (0.4, 0.4, 0.2)$$

**Reducció 3:** Finalment, substituirem a $p^2$ els dos últims valors 0.4, 0.2 per la seva suma, obtenint d'aquesta manera

$$p^3 = (0.4, 0.6)$$

Ara, per la distribució $p^3$ formarem l'arbre de codificació trivial de Huffman (que anomenarem $A_3$), que consisteix en 3 vèrtexs, un que és el vèrtex arrel i dos més que són fulles. Etiquetarem les dues fulles d'$A_3$ amb "0.4" i "0.6" respectivament, i el vèrtex arrel amb "1"; l'arbre $A_3$ està format pels vèrtexs A, B, C de la figura 2.

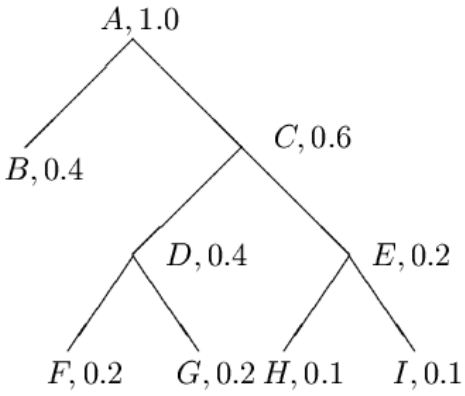


**Figura 2 - Arbre de codificació de Huffman A per (0.4, 0.2, 0.2, 0.1, 0.1)**

Veiem que al anar de $p^2$ a $p^3$ (veure Reducció 3), el component 0.6 de $p^3$ va ser format sumant 0.4 i 0.2. Això ens diu que podem formar l'arbre de codificació de Huffman $A_2$ per a $p^2$ fent créixer des de la fulla amb l'etiqueta 0.6 (vèrtex C, figura 2) dues noves fulles etiquetades 0.4 i 0.2 respectivament: l'arbre $A_2$ que consisteix en el vèrtexs A, B, C, D i E. De la mateixa manera, podem crear l'arbre $A_1$ fent créixer les fulles 0.2 i 0.2 des de la fulla 0.4 ($A_1$ consta dels vèrtexs A, B, C, D, E, F i G) , i l'arbre $A_0$ fent créixer les fulles 0.1 i 0.1 a partir de la fulla 0.2., que es l'arbre sencer de la figura 2.

Es pot veure fàcilment que el nostre arbre de codificació de Huffman indueix un codi compacte (1, 3, 3, 3, 3) – recordem que el nombre de bits coincideix amb el nombre de

branques que hem de travessar fins arribar des del vèrtex principal fins al vèrtex que correspon al símbol que volem codificar. El factor de compressió d'aquest codi és

$$(1, 3, 3, 3, 3)\cdot(0.4, 0.2, 0.2, 0.1, 0.1) = 2{,}2 \text{ bits de codi/mostra de dades}$$

*NOTA: L'operador '·' es el producte component a component*

(A l'hora de fer el càlcul anterior, ens hem d'assegurar que els components del vector de longitud de codi estan en ordre *ascendent*, i que els components del vector de probabilitats estan en ordre *descendent*)

***Exercici 1.*** **Es pot verificar fàcilment que**

$$\mathbf{(2, 2, 2, 3, 3)\cdot(0.4, 0.2, 0.2, 0.1, 0.1) = 2.2}$$

**i que**

$$\mathbf{(1, 2, 3, 4, 4)\cdot(0.4, 0.2, 0.2, 0.1, 0.1) = 2.2.}$$

**Per tant, tant el codi compacte (2, 2, 2, 3, 3) com el (1, 2, 3, 4, 4) són també codis de Huffman per la distribució de probabilitats (0.4, 0.2, 0.2, 0.1, 0.1) (perquè donen el mateix factor de compressió que el codi de Huffman que hem trobat abans). Aplica un altre cop l'algorisme de Huffman, però de manera que el codi resultant sigui ara el codi compacte (2, 2, 2, 3, 3). Després, fes el mateix però que ara surti el codi (1, 2, 3, 4, 4). La distribució de probabilitats (0.4, 0.2, 0.2, 0.1, 0.1) és interessant en el sentit de que es poden dissenyar tres codis de Huffman diferents per ella. Per tant, veiem que el codi de Huffman no té per què ser únic. Només serà únic per una classe particular de distribucions de probabilitats. Per la major part de distribucions, sempre n'hi haurà més d'una possibilitat.**

## Càlcul de la longitud de codi mitjançant MATLAB

No és difícil fer una funció MATLAB que calculi la longitud del codi de Huffman per un vector de probabilitats donat. Per poder realitzar els exemples posteriors, crea el fitxer "huffmancodelengths.m" al teu directori de treball de MATLAB, amb el següent codi:

```
function L = huffmancodelengths(p)
%p es el vector de probabilitats, L = huffmancodelengths(p)
%es el vector de longituds de codis pel codi Huffman code per a p
%en ORDRE DECREIXENT. Com que p esta sense ordenar, l'esperança
%de longitud de codi es troba executant "sum(sort(p).*L)" després
%de calcular L
k=length(p);
M=zeros(k,k+1);
M(:,1)=p';
R=k;
while R > 1
v=M(:,1);
[u,j]=sort(v);
j1=min(j(1),j(2));
j2=max(j(1),j(2));
r1=M(j1,:);
```

```
r2=M(j2,:);
t1=find(r1==0);
t2=find(r2==0);
m1=min(t1);
m2=min(t2);
if m1 > 2
m1=m1-1;
end
if m2 > 2
m2=m2-1;
end
v1=r1(2:m1);
v2=r2(2:m2);
bottom=[r1(1)+r2(1) v1+1 v2+1 zeros(1,k+2-m1-m2)];
N=zeros(R-1,k+1);
for i=1:k+1;
s1=1:j1-1;
s2=j1+1:j2-1;
s3=j2+1:R;
Q=M(:,i)';
Qnew=[Q(s1) Q(s2) Q(s3) bottom(i)];
N(:,i)=Qnew';
end
R=R-1;
M=N;
end
z=M(1,2:k+1);
y=sort(z);
L=y(length(y):-1:1);
```

Veiem un parell d'exemples:

*Exemple 2.* Verifiquem amb la nostra nova funció MATLAB que el resultat de l'exercici 1 és correcte.

```
L=huffmancodelengths([.4 .2 .2 .1 .1])
L =
3 3 2 2 2
%Les longituds estan en ordre decreixent, i les volem
%en ordre creixent
sort(L)
ans =
2 2 2 3 3
%Per tant, un codi Huffman és el codi compacte (2,2,2,3,3)
```

De l'exemple anterior veiem que la nostra funció només dona un dels possibles codis de Huffman per la distribució de probabilitats donada, no busca solucions alternatives.

*Exemple 3.* Aquest exemple il·lustra l'ús de la funció *huffmancodelengths* sobre un conjunt de dades real. Mirarem quin és el factor de compressió que obtenim codificant mitjançant Huffman els píxels de la imatge de Lena (un clàssic del processament d'Imatge, veure figura 3) . Recordem que Huffman és una codificació píxel a píxel:

**Figura 3 - Imatge de na Lena, 512x512. Un clàssic!!!**

```
A=imread('lena_512.gif');
colormap(gray(256));
image(A) %Comprovem que realment es la Lena ;)
A=reshape(A,[1 512*512]);
freq=hist(A,0:255);
bar(freq) %Aquest es l'histograma
freq=freq((find(freq>0))); %Nomes els pixels presents a la imatge
length(freq)
ans =
236
%236 dels 256 valors de píxel realment apareixes a Lena
p=freq/sum(freq) %p es la distribucio empirica de primer ordre
L=huffmancodelengths(p);
factor_compressio=sum(sort(p).*L)
factor_compressio =
7.6241
```

Podem concloure que codificar la Lena mitjançant Huffman no és molt útil. Descomprimit, la Lena té 8 bits/píxel, i el resultat de Huffman és de 7.6241, una millora bastant modesta. N'hi ha altres aproximacions, com a la codificació diferencial, que ens

permeten un factor de compressió molt millor, i sense pèrdues. Els mètodes més sofisticats tenen en compte la correlació entre els píxels veïns, una codificació píxel a píxel no pot aconseguir això.

***Exercici 2.*** **Calcula la longitud de codi de Huffman pels fitxers de la sessió anterior i comenta els resultants.**

## Codis de Shannon-Fano

El problema de l'algorisme de Huffman, que acabem de veure, és que no es un algorisme de temps lineal, amb la qual cosa per alfabets grans pot arribar a trigar molt de temps.
El codi de Shannon-Fano, que veurem a aquesta última secció, alleuja aquest problema. Al contrari que passava al codi de Huffman, a un codi de Shannon-Fano coneixem la longitud dels codis per avançat, i, com veurem, hi ha un mètode molt ràpid per trobar qualsevol codi de Shannon-Fano. La part dolenta és que és sub-òptim, es a dir, és un codi no compacte, i per tant no pot ser òptim. Malgrat això, en molts casos el codi de Shannon-Fano té un rendiment no massa lluny del de Huffman, per això per determinats casos és més que suficient.

El codi de Shannon-Fano assigna a cada lletra $s_i$ de l'alfabet, un codi binari de longitud

$$L_i = \left[ -\log_2 p(s_i) \right] \quad (3)$$

Il·lustrarem el mètode de construcció del codi de Shannon-Fano mitjançant un exemple senzill. Suposem que el nostre alfabet de tamany N=8. Les probabilitats associades als 8 símbols les trobem a la segona columna de l'esquerra de la taula 1. Fixem-nos que hem llistat les probabilitats de més gran a més petit – ha de ser així perquè el mètode funcioni. Les longituds de codi $L_i$ les hem calculat aplicant l'equació (3), i es troben a la segona columna de la dreta. Els codis de la columna més a la dreta s'obtenen expandint les probabilitats acumulades F(si) de la columna central al nombre de dígits requerit per la longitud de codi, sobre la seva expansió binària (els codis són els dígits en negreta de les expansions binàries, o sigui, **0**, **010**, **100**, **1010**, **1100**, **11011**, **11100**, **11110**).

**Taula 1 – Exemple de codificació de Shannon-Fano**

| Símbol | $p(s_i)$ | $F(s_i) = \sum_{k<i} p(s_i)$ | $L_i$ | Codi (expansió de $F(s_i)$) |
|---|---|---|---|---|
| $s_1$ | 0.3 | 0 | 2 | .**00**000... |
| $s_2$ | 0.2 | 0.3 | 3 | .**010**01... |
| $s_3$ | 0.15 | 0.5 | 3 | .**100**00... |
| $s_4$ | 0.1 | 0.65 | 4 | .**1010**01... |
| $s_5$ | 0.1 | 0.75 | 4 | .**1100**00... |
| $s_6$ | 0.05 | 0.85 | 5 | .**11011**00... |
| $s_7$ | 0.05 | 0.90 | 5 | .**11100**11... |
| $s_8$ | 0.05 | 0.95 | 5 | .**11110**11... |

Fem un exemple del procediment de formació de la paraula codi  demostrant que el codi pel símbol $s_6$ es realment (1, 1, 0, 1, 1). Primer de tot, expandim la probabilitat 0.85 en la seva expansió binària, que té la següent forma:

$$0.85 = b_1 / 2 + b_2 / 4 + b_3 / 8 + b_4 / 16 + b_5 / 32 + \sum_{i=6}^{\infty} (b_i / 2^i) \quad (4)$$

essent $b_i$ el dígit binari número i després del punt decimal a l'expansió binària de 0.85, i pot valer 0 ó 1. Hem de resoldre l'equació (4) per $b_1$, $b_2$, $b_3$, $b_4$ i $b_5$, perquè el codi ha de ser de longitud 5, i aquests 5 bits formen la paraula codi. Si multipliquem els dos cantons de l'equació (4) per 32, obtenim

$$32 \cdot (0.85) = 16b_1 + 8b_2 + 4b_3 + 2b_4 + b_5 + x\,,$$

a on $x$ es una fracció entre 0 i 1. Arrodonim les dues bandes de l'equació precedent a sencer, menyspreant per tant la part fraccional $x$:

$$27 = \lfloor 32 \cdot (0.85) \rfloor = 16b_1 + 8b_2 + 4b_3 + 2b_4 + b_5\,.$$

Per tant ($b_1$, $b_2$, $b_3$, $b_4$, $b_5$) és l'expansió binària del sencer 27, la qual és

$$(b_1, b_2, b_3, b_4, b_5) = (1, 1, 0, 1, 1),$$

com volíem demostrar. Només com a comprovació, calculem

$$(16, 8, 4, 2, 1)\cdot(1, 1, 0, 1, 1) = 27.$$

***Exercici 3.*** **Fer una funció MATLAB que ens doni la longitud de codi de Shannon-Fano d'un vector de probabilitats. Aplica aquesta funció als diferents fitxers que vam tractar l'altre dia i comenta els resultats.**

***Exercici 4.*** **Compara els resultats d'Entropia, longitud de codi de Huffman i longitud de codi de Shannon-Fano. Quines conclusions pots treure?**

***Exercici 5.*** **Fes un programa que calculi les paraules de codi de Shannon-Rao per un fitxer de dades. El programa es pot presentar en MATLAB o en C.**