Códigos de Huffman

Fernando Martínez
fernando.martinez@upc.edu

Departament de Matemàtiques • Universitat Politècnica de Catalunya

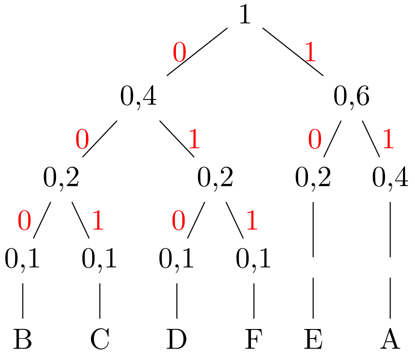
22 de febrero de 2022

Capítulo 3 de Introduction to Data Compression **Sayood, Khalid** Morgan Kaufmann, 2012, 4th ed.

Algoritmo

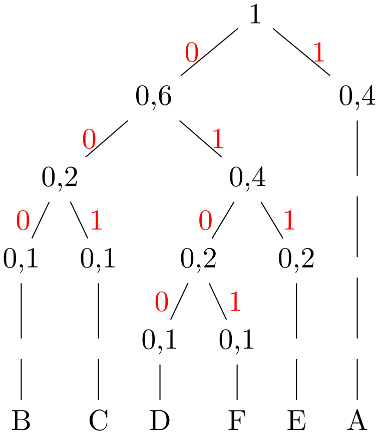
Ejemplo

(A, 0,4), (B, 0,1), (C, 0,1), (D, 0,1), (E, 0,2), (F, 0,1)



A$\leftrightarrow$11, B$\leftrightarrow$000, C$\leftrightarrow$001, D$\leftrightarrow$010, E$\leftrightarrow$10, F$\leftrightarrow$011 $\tilde{l} = 2,4$ bits/símbolo.

Otra posibilidad:



A↔1, B↔000, C↔001, D↔0100, E↔011, F↔0101 $\tilde{l} = 2,4$ bits/símb.

Observaciones:

I) Los códigos de Huffman no son únicos.

II) Por construcción, son prefijos.

III) Si $q > 2$ en la primera iteración se han de agrupar t símbolos: $2 \leq t \leq q,\ t \equiv n \mod q-1$. Veámoslo:

1. si $q = 3$ en cada iteración hay 1, 3, 5, 7, 9... vértices
2. si $q = 4$ en cada iteración hay 1, 4, 7, 10, 13... vértices
3. si $q = 5$ en cada iteración hay 1, 5, 9, 13, 17... vértices

en cada iteración se ha de cumplir que el número de vértices restantes k sea $k \equiv 1 \mod q-1$. Si inicialmente hay n vértices y agrupamos t apareciendo un nuevo vértice se ha de cumplir $k = n - t + 1 \equiv 1 \mod q-1$, por lo tanto $t \equiv n \mod q-1$.

IV) Las longitudes de las t palabras más largas son iguales.

V) Las t palabras más largas difieren en la última letra.

Teorema

Los códigos de Huffman tienen longitud media mínima.

Demostración

Observaciones previas:

Supongamos $p_1 \geq p_2 \geq \cdots \geq p_n$.

Sea $\mathcal{C}_H$ código de Huffman con palabras de longitudes $l_1 \leq l_2 \leq \cdots \leq l_{n-1} = l_n$.

Sea $\mathcal{C} = \{c_1, ..., c_n\}$ código con palabras de longitudes $m_1 \leq m_2 \leq \cdots \leq m_{n-1} = m_n$.

- Si las longitudes no están ordenadas, reordenándolas tenemos un código con $\tilde{l}$ menor,
- ni $m_{n-1} < m_n$ podemos eliminar letras de c_n hasta que tenga la misma longitud que c_{n-1} (prefijo) resultando un código con $\tilde{l}$ menor.

...

$\tilde{l}(\mathcal{C}_H) \leq \tilde{l}(\mathcal{C})$ por inducción sobre n, número de palabras del código:

1. $n = 1$, $\tilde{l}(\mathcal{C}_H) = 1 \leq \tilde{l}(\mathcal{C})$ ✓
2. Cierto hasta $n-1$ $\Rightarrow$ cierto para n.

   Sea $\mathcal{C}_H$ código de Huffman para la fuente $\mathcal{S}$: $p_1 \geq p_2 \geq \cdots \geq p_n$

   Sea $\bar{\mathcal{C}}_H$ código de Huffman para la fuente $\bar{\mathcal{S}}$: $p_1 \geq p_2 \geq \cdots \geq p_{n-2}$, $p_{n-1} + p_n$

   Entonces:

   $\tilde{l}(\mathcal{C}_H) = p_1 l_1 + \cdots + p_{n-1} l_{n-1} + p_n l_n \overset{l_{n-1}=l_n}{=} p_1 l_1 + \cdots + (p_{n-1} + p_n) l_n$

   $\tilde{l}(\bar{\mathcal{C}}_H) = p_1 l_1 + \cdots + p_{n-2} l_{n-2} + (p_{n-1} + p_n)[l_n - 1]$ (por construc. de Huffman).

   Comparando $\tilde{l}(\bar{\mathcal{C}}_H) = \tilde{l}(\mathcal{C}_H) - (p_{n-1} + p_n)$.

   Sea ahora $\mathcal{C}$ un código para la fuente $\mathcal{S}$ y $\bar{\mathcal{C}}$ un código para la fuente $\bar{\mathcal{S}}$ obtenido a partir de $\mathcal{C}$ eliminando c_{n-1} y c_n y añadiendo $\tilde{c}_{n-1}$ construida eliminando la última letra de c_n (o de c_{n-1}).

   Entonces $\tilde{l}(\bar{\mathcal{C}}) = \tilde{l}(\mathcal{C}) - (p_{n-1} + p_n)$

   Por H.I. $\tilde{l}(\bar{\mathcal{C}}) \geq \tilde{l}(\bar{\mathcal{C}}_H)$, entonces $\tilde{l}(\mathcal{C}) - (p_{n-1} + p_n) \geq \tilde{l}(\mathcal{C}_H) - (p_{n-1} + p_n)$ y

   $$\tilde{l}(\mathcal{C}) \geq \tilde{l}(\mathcal{C}_H). ✓$$

□

Ver `02_Huffman.ipynb`

Teorema de la codificación sin ruido

Lema (Gibbs)

Sean:

$$p_1, ...p_n \text{ reales tales que } p_i > 0 \text{ con } \sum p_i = 1,$$

$$x_1, ...x_n \text{ reales tales que } x_i > 0 \text{ con } \sum x_i \leq 1.$$

Entonces:

$$\sum p_i \log \frac{1}{p_i} \leq \sum p_i \log \frac{1}{x_i}.$$

Demostración.

Fijemos p_i y definamos $f : \mathbb{R}^n \longrightarrow \mathbb{R}$, $f(x_1, ..., x_n) = \sum \left(p_i \log \frac{1}{x_i} - p_i \log \frac{1}{p_i}\right)$.
Si buscamos el mínimo de f con la condición $\sum x_i \leq 1$ fácilmente obtenemos que se alcanza cuando $x_i = p_i$. □

Teorema (Primer teorema de Shannon – codificación sin ruido, V1)

Sea $\mathcal{C}_H$ un código de Huffman para una fuente $\mathcal{S}$, entonces

$$H(\mathcal{S}) \leq \tilde{l}(\mathcal{C}_H) < H(\mathcal{S}) + 1.$$

Demostración.

$p_1 \geq p_2 \geq ... \geq p_n > 0$. $\mathcal{C}_H = \{c_1, ...c_n\}$ cumple la desigualdad de Kraft-McMillan, $\sum 2^{-l_i} \leq 1$.

- Primera desigualdad:

$$H(\mathcal{S}) = \sum p_i \log \frac{1}{p_i} \overset{Gibbs}{\leq} \sum p_i \log \frac{1}{2^{-l_i}} = \sum p_i l_i = \tilde{l}(\mathcal{C}_H)$$

- Segunda desigualdad: Definimos $a_i \equiv \lceil \log \frac{1}{p_i} \rceil \geq \log \frac{1}{p_i} > 0$ i.e. $a_i \geq 1$.

  Además $-a_i \leq \log p_i \Rightarrow 2^{-a_i} \leq p_i \Rightarrow \sum 2^{-a_i} \leq \sum p_i = 1$. Por el teorema de Kraft, existe un código $\mathcal{C}$ con palabras de longitudes $a_1, ..., a_n$. Entonces

$$\tilde{l}(\mathcal{C}_H) \leq \tilde{l}(\mathcal{C}) = \sum a_i p_i \overset{\text{def. } a_i}{<} \sum p_i \left(\log \frac{1}{p_i} + 1 \right) = H(\mathcal{S}) + 1.$$

$\square$

Ejemplo

$\mathcal{S} = \{(s_1, 1/4), (s_2, 3/4)\}$
Para esta fuente $H(\mathcal{S}) = 0{,}811$.

- Si codificamos cada símbolo:
  - $s_1 \to 0$,
  - $s_2 \to 1$,

  $\tilde{l} = 1$ bit/símbolo.
- Si codificamos pares de símbolos:
  - $s_1 s_1 \to 010$, $(1/16)$
  - $s_1 s_2 \to 011$, $(3/16)$
  - $s_2 s_1 \to 00$, $(3/16)$
  - $s_2 s_2 \to 1$, $(9/16)$

  $\tilde{l} = 1{,}688$ bits/2 símbolos; $0{,}844$ bits/símbolo

...

Ejemplo (cont.)

- Si codificamos tríos de símbolos:
  - $s_1s_1s_1 \to 11100$, $(1/64)$
  - $s_1s_1s_2 \to 11101$, $(3/64)$
  - $s_1s_2s_1 \to 11110$, $(3/64)$
  - $s_2s_1s_1 \to 11111$, $(3/64)$
  - $s_2s_2s_1 \to 100$, $(9/64)$
  - $s_2s_1s_2 \to 1010$, $(9/64)$
  - $s_1s_2s_2 \to 110$, $(9/64)$
  - $s_2s_2s_2 \to 0$, $(27/64)$

$\tilde{l} = 2{,}469$ bits/3 símbolos; $0{,}823$ bits/símbolo

En este ejemplo observamos que podemos disminuir el número de bits por símbolos si incrementamos el número de símbolos codificados a la vez.

Podemos preguntarnos:

i) ¿Hay límite al # bits/símbolo?
ii) ¿Cual es?
iii) ¿Vale la pena?

Definición

Extensión k-ésima de la fuente $\mathcal{S} = (S, \mathcal{P})$: $\mathcal{S}^k = (S^k, \mathcal{P}^k)$ siendo:

$$S^k = S \times S \times \cdots \times S,$$

$$s \in S^k, \quad s = s_{i_1} s_{i_2} ... s_{i_k}, \quad s_{i_j} \in S,$$

$$\mathcal{P}^k(s) = \mathcal{P}(s_{i_1})\mathcal{P}(s_{i_2}) \cdots \mathcal{P}(s_{i_k}).$$

Teorema

Sea $\mathcal{S}$ una fuente y $\mathcal{S}^k$ su extensión k-ésima, entonces

$$H(\mathcal{S}^k) = k\, H(\mathcal{S})$$

Demostración.

$$H(\mathcal{S}^k) = \sum_{i_1} \sum_{i_2} \cdots \sum_{i_k} p_{i_1} p_{i_2} \cdots p_{i_k} \log \frac{1}{p_{i_1} p_{i_2} \cdots p_{i_k}}$$

$$= \sum_{i_1} \cdots \sum_{i_k} p_{i_1} \cdots p_{i_k} \log \frac{1}{p_{i_1}} + ... + \sum_{i_1} \cdots \sum_{i_k} p_{i_1} \cdots p_{i_k} \log \frac{1}{p_{i_k}}$$

Notemos que:
$\sum_{i_1} \cdots \sum_{i_k} p_{i_1} \cdots p_{i_k} \log \frac{1}{p_{i_1}} = \left(\sum_{i_1} p_{i_1} \log \frac{1}{p_{i_1}}\right) \left(\sum_{i_2} p_{i_2}\right) \cdots \left(\sum_{i_k} p_{i_k}\right) = H(\mathcal{S}) \cdot 1 \cdots 1.$

Por lo tanto:

$$H(\mathcal{S}^k) = H(\mathcal{S}) + \cdots + H(\mathcal{S}) = k\, H(\mathcal{S}).$$

$\square$

Observaciones:

1. La entropía por símbolo no cambia $H(\mathcal{S}^k)/k = H(\mathcal{S})$.
2. Estamos suponiendo una fuente sin memoria, o sea que la aprarición de un símbilo no condiciona la probabilidad de aparición del siguiente símbolo. No siempre ocurre, por ejemplo, si estamos leyendo un texto y parece q, la próxima letra será, probablemente, u.

Teorema (Primer teorema de Shannon – codificación sin ruido, V2)

Sea $\mathcal{C}_K$ *un código de Huffman para la extensión* k*-ésima de la fuente* $\mathcal{S}$. *Entonces:*

$$\lim_{k\to\infty} \frac{\tilde{l}(\mathcal{C}_K)}{k} = H(\mathcal{S}).$$

Demostración.

$$k\,H(\mathcal{S}) = H(\mathcal{S}^k) \leq \tilde{l}(\mathcal{C}_K) < H(\mathcal{S}^k) + 1 = k\,H(\mathcal{S}) + 1.$$

Por lo tanto

$$H(\mathcal{S}) \leq \frac{\tilde{l}(\mathcal{C}_K)}{k} < H(\mathcal{S}) + \frac{1}{k}.$$

Tomando límite $k \to \infty$ obtenemos el resultado deseado. $\square$

Observaciones:

1. La entropía marca un límite a la compresión sin pérdidas.
2. Nos podemos acercar tanto como queramos al límite extendiendo la fuente, pero la convergencia puede ser muy lenta.
3. Aunque matemáticamente está claro qué es la entropía una vez definido el modelo, el problema reside en modelizar la fuente. De ahí la existencia de distintos métodos de compresión y de sus resultados diferentes.

`Ver en 02.1_Huffman.py cómo varía la ratio de compresión de La Regenta si tomamos frecuencias para agrupaciones de 1, 2, 3... símbolos.`

Códigos adaptativos

A la hora de construir el código se presuponen conocidas las frecuencias, pero no siempre es así. Es conveniente adaptar las probabilidades a la situación concreta:

- Leer el fichero dos veces, una para calcular las frecuencias y otra para codificarlo una vez construido el código. Este sería un método semiadaptativo. Hay que enviar el código.
- Ir construyendo el código a medida que se lee el fichero. En este caso no es necesario enviar el código. Permite empezar a codificar sin necesidad de leer todo el mensaje y *reinicializar* el código usado a mitad de la lectura/transmisión cuando la ddp de los símbolos varía considerablemente.

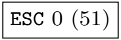
Método adaptativo. Inicialmente el árbol sólo tiene un nodo que representa `ESC`:

- Si se lee un símbolo que no está en el árbol se escribe el código correspondiente a `ESC` y el símbolo tal cual (ASCII, UTF8,...)
- Si se lee un símbolo que está en el árbol se escribe el código correspondiente al símbolo y se actualiza el árbol.*

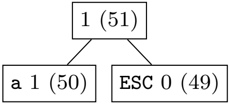
*Para actualizar el árbol de forma eficiente se le pide que conserve la *sibling property*: Se numeran los vértices de forma que V_{2k-1} y V_{2k} tengan el mismo padre y que el índice del padre sea mayor que el de los hijos, o sea que el padre sea V_{2k+1+i}, $i \geq 0$. Además pedimos que los pesos asociados a los vértices estén ordenados según su numeración $\omega_i \leq \omega_{i+1}$ siendo ω_i el peso asociadoal vértice V_i.

Ejemplo

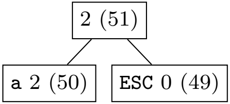
Texto a codificar: **aardvark** (necesitamos 2*26-1 nodos)[a]
Árbol Inicial:



- Leemos **a**, no está en el árbol, enviamos el código ASCII de **a**: 0x61 = 01100001.
  Mensaje 01100001
  Actualizamos el árbol:

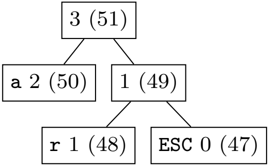


- Leemos **a**, está en el árbol, enviamos el código de **a**: 0.
  Mensaje 011000010
  Actualizamos el árbol:

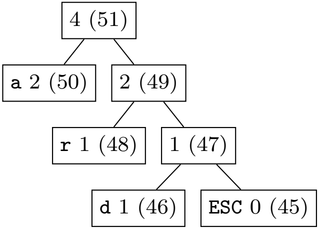


[a] (Ver páginas 69–71 de la 4ª edición Sayood)

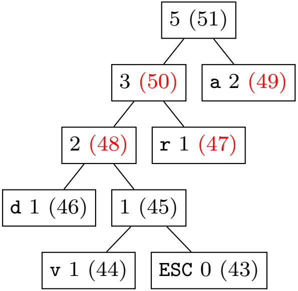
- Leemos `r`, no está en el árbol, enviamos el código de `ESC`: 1 y el código ASCII de `r`: 0x72 = 01110010.
  Mensaje 011000010 1 01110010
  Actualizamos el árbol:



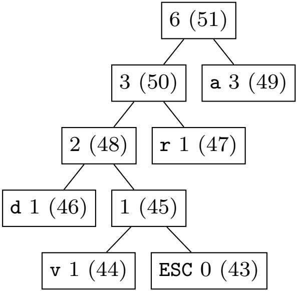
- Leemos `d`, no está en el árbol, enviamos el código de `ESC`: 11 y enviamos el código ASCII de `d`: 0x64 = 01100100.
  Mensaje 011000010101110010 11 01100100
  Actualizamos el árbol:



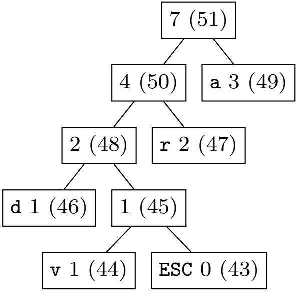
- Leemos `v`, no está en el árbol, enviamos el código de `ESC`: 111 y enviamos el código ASCII de `v`: 0x76 = 01110110.
  Mensaje 0110000101011100101101100100 111 01110110
  Actualizamos el árbol:



- Leemos `a`, está en el árbol, enviamos el código de `a`: 1.
  Mensaje 011000010101110010110110010011101110110 1
  Actualizamos el árbol:



- Leemos **r**, está en el árbol, enviamos el código de **r**: 01.
  Mensaje 01100001010111001011011001001110111101101 01
  Actualizamos el árbol:



Ejemplo: Recibimos 011000010101110010110110010011101110110101.

- Los 8 primeros bits 011000010101110010110110010011101110110101 corresponden a `a`. Mensaje `a`. Construimos el árbol 4.
- Leemos 011000010101110010110110010011101110110101 que en el árbol corresponde a `a`. Mensaje `aa`. Actualizamos el árbol 4.
- Leemos 011000010101110010110110010011101110110101 que en el árbol corresponde a `ESC`, los 8 siguientes bits 011000010101110010110110010011101110110101 corresponden a `r`. Mensaje `aar`. Actualizamos el árbol 21.
- Leemos 011000010101110010110110010011101110110101 que en el árbol corresponde a `ESC`, los 8 siguientes bits 011000010101110010110110010011101110110101 corresponden a `d`. Mensaje `aard`. Actualizamos el árbol 21.
- Leemos 011000010101110010110110010011101110110101 que en el árbol corresponde a `ESC`, los 8 siguientes bits 011000010101110010110110010011101110110101 corresponden a `v`. Mensaje `aardv`. Actualizamos el árbol 22.
- Leemos 011000010101110010110110010011101110110101 que en el árbol corresponde a `a`. Mensaje `aardva`. Actualizamos el árbol 23.
- Leemos 011000010101110010110110010011101110110101 que en el árbol corresponde a `r`. Mensaje `aardvar`. Actualizamos el árbol 24 ...

Códigos de Shannon-Fao

1. Se ordenan los símbolos según sus probabilidades.
2. Se dividen en dos subconjuntos de parecidas probabilidades, al primer subconjunto se les asigna 0 y al segundo 1.

Para cada subconjunto se procede de forma análoga hasta que todos los subconjuntos tengan un único elemento.

Ejemplo: Fuente con ddp $\{0{,}20, 0{,}15, 0{,}25, 0{,}05, 0{,}1, 0{,}15, 0{,}1\}$.

0,25	0	0			00
0,20	0	1			01
0,15	1	0	0		100
0,15	1	0	1		101
0,1	1	1	0		110
0,1	1	1	1	0	1110
0,05	1	1	1	1	1111

$\tilde{l} = 2{,}7$ bits/símbolo, en este caso es la misma que Huffman (aunque en muchos casos pueda coincidir con la de un código de Huffman no siempre es así).