# **Tema:** Árboles de Huffman

**Asignatura:** Métodos Algorítmicos de Resolución de Problemas. **Profesor:** Ricardo Peña

Curso 2013/14

## 1. Compresión de ficheros

⋆ Dado un fichero con cualquier tipo de contenido (texto plano, código ejecutable, imagen, sonido, etc.), siempre es posible descomponerlo en unidades de longitud fija, por ejemplo octetos, si bien los razonamientos que siguen son aplicables a cualquier otra unidad.

⋆ Las frecuencias de los octetos serán en general distintas. Sea $C$ el conjunto de todos los octetos distintos y $f_c$ la frecuencia del octeto $c \in C$. Entonces, la longitud total del fichero expresada en bits viene dada por la fórmula:

$$L = \sum_{c \in C} 8 \times f_c$$

⋆ Una estrategia para disminuir el tamaño del fichero sin perder información consiste en abandonar la idea de longitud fija y codificar los octetos mediante cadenas de bits de *longitud variable*: a los octetos más frecuentes se les asigna códigos más cortos, y a los menos frecuentes códigos más largos. Los octetos ausentes no recibirían ningún código.

⋆ Si $l_c$ es el número de bits de la codificación asignada al octeto $c$, la longitud total del fichero viene dada ahora por:

$$L' = \sum_{c \in C} l_c \times f_c$$

Sin pérdida de generalidad, llamaremos *carácter* a cada uno de los octetos $c \in C$.

⋆ La función de compresión calcularía las fecuencias $f_c$, después calcularía la codificación óptima, y traduciría cada carácter a su código de longitud variable, produciendo el fichero comprimido. Para que la descompresión sea posible, son necesarias dos cosas:

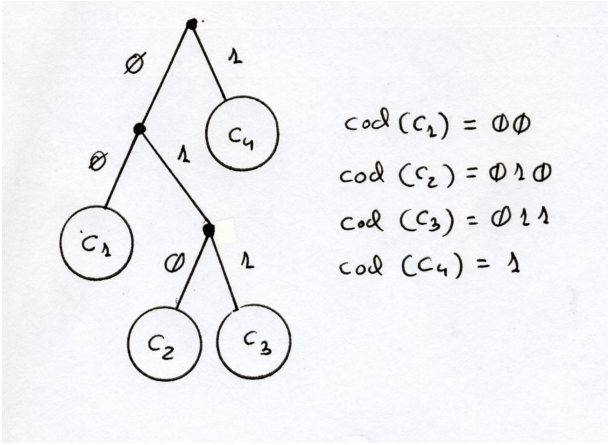
1. Incluir el diccionario de codificación al principio del fichero comprimido
2. Que los códigos asignados satisfagan la *propiedad de prefijo*: no deben existir dos caracteres tales que el código del primero sea un prefijo del código del segundo. Si existieran, la descodificación se encontraría con situaciones ambíguas.

⋆ El estudiante de doctorado del MIT David A. Huffman recibió en 1952 el encargo de su director de calcular a partir de las frecuencias $f_c$ una codificación *óptima*, es decir aquella que arrojaría una longitud $L'$ mínima del fichero comprimido.

⋆ Huffman ideó un tipo particular de árboles binarios para organizar una codificación cualquiera de longitud variable que satisfaga la propiedad de prefijo. Un *árbol de Huffman*:

- O bien es una hoja que contiene un carácter,
- O bien es la unión ordenada de dos árboles de Huffman. El árbol izquierdo añade un 0 al código de longitud variable, mientras que el árbol derecho añade un 1.

Debe haber una hoja por cada $c \in C$. El código asignado a $c$ se obtiene siguiendo la cadena de bits desde la raíz del árbol de Huffman hasta la hoja que contiene a $c$.




⋆ El problema se puede formular entonces como, dadas las frecuencias $\{f_c \mid c \in C\}$, calcular un árbol de Huffman $H$ para $C$ tal que $L = \sum_{c \in C} l_c \times f_c$ sea mínima.

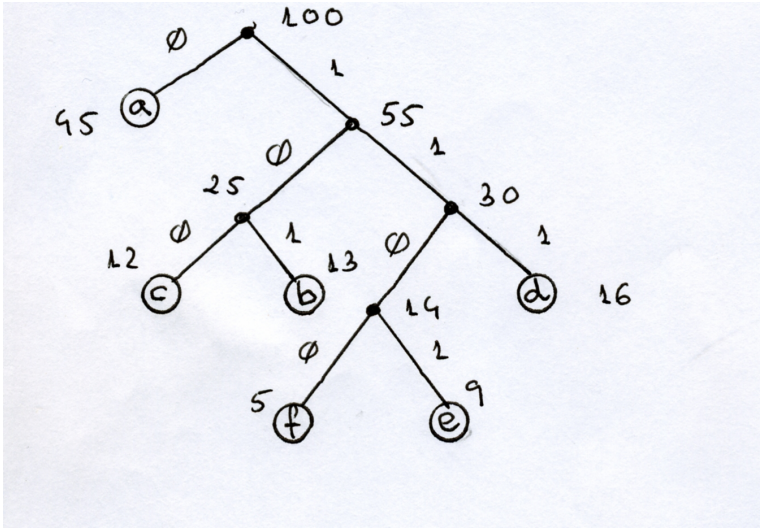
⋆ *Ejemplo:* Supongamos $C = \{a, b, c, d, e, f\}$ y la siguiente tabla de frecuencias:

| carácter | frec. en miles |
|---|---|
| $a$ | 45 |
| $b$ | 13 |
| $c$ | 12 |
| $d$ | 16 |
| $e$ | 9 |
| $f$ | 5 |

Con un código de longitud fija, son necesarios al menos tres bits para codificar seis caracteres, lo que da una longitud del fichero de 300.000 bits:

$$L = 3 \times 1000 \times (45 + 13 + 12 + 16 + 9 + 5) = 300.000$$

El árbol calculado por Huffman es en este caso:




Es decir, $l_a = 1,\ l_b = l_c = l_d = 3,\ l_e = l_f = 4$. La longitud del fichero comprimido es entonces:

$$L = 1000 \times (45 \times 1 + (13 + 12 + 16) \times 3 + (9 + 5) \times 4) = 224.000$$

un $25,4\,\%$ más pequeño que el fichero original.

⋆ Partiendo del árbol de Huffman $H$, la descodificación del fichero comprimido procedería del modo siguiente:

1. Recorrer los bits iniciales del fichero comprimido siguiendo las ramas correspondientes de $H$ hasta llegar a una hoja.
2. Copiar el carácter contenido en la hoja al fichero descomprimido.
3. Eliminar del fichero los bits recorridos (o avanzar el cursor del mismo)
4. Si el fichero remanente es vacío, terminar. En caso contrario, volver a (1).

## 2. Algoritmo voraz para calcular el árbol de Huffman óptimo

⋆ El algoritmo ideado por Huffman mantiene una estructura ordenada por frecuencias crecientes que contiene árboles de Huffman. Inicialmente contiene sólo las hojas, cada una con un cáracter y su frecuencia, ordenada por frecuencias crecientes.

⋆ En cada iteración, elimina de la estructura los dos árboles con menor frecuencia y forma con ellos el árbol resultante de su unión. La frecuencia asignada a dicho aŕbol es la suma de las frecuencias de los árboles que une. Inserta el árbol y su frecuencia en la estructura.

⋆ Cuando sólo queda un árbol en la estructura, ese es el árbol óptimo.

⋆ La estructura apropiada es un *montículo de mínimos*, cuyos elementos son tuplas formadas por un árbol de Huffman y un entero que representa su frecuencia.

⋆ *Ejemplo:* Para el fichero mencionado más arriba, se muestran las secuencias ordenadas por las que pasa el algoritmo. Denotamos por $h(c)$ la hoja que contiene el carácter $c$ y por $h_1 \bullet h_2$ el árbol unión de $h_1$ y $h_2$.

| | |
|---|---|
| 0 | $[(h(f),5),\ (h(e),9),\ (h(c),12),\ (h(b),13),\ (h(d),16),\ (h(a),45)]$ |
| 1 | $[(h(c),12),\ (h(b),13),\ (h(f)\bullet h(e),14),\ (h(d),16),\ (h(a),45)]$ |
| 2 | $[(h(f)\bullet h(e),14),\ (h(d),16),\ (h(c)\bullet h(b),25),\ (h(a),45)]$ |
| 3 | $[(h(c)\bullet h(b),25),\ ((h(f)\bullet h(e))\bullet h(d),30),\ (h(a),45)]$ |
| 4 | $[(h(a),45),\ ((h(c)\bullet h(b))\bullet((h(f)\bullet h(e))\bullet h(d)),55)]$ |
| 5 | $[(h(a)\bullet((h(c)\bullet h(b))\bullet((h(f)\bullet h(e))\bullet h(d))),100)]$ |

⋆ Pseudocódigo del algoritmo de Huffman:

```
1  function Huffman (C:caracteres, f:tablaFrecuencias) return H:arbolHuffman
2  var Q:colaDeMinimos
3    n = |C|
4    for c in C
5      do Q = insertar(Q, (hoja(c),f(c)))
6    for i = 1 to n-1
7      do (h1,f1) = min(Q); Q = borraMin(Q)
8         (h2,f2) = min(Q); Q = borraMin(Q)
9         Q = insertar(Q, (unir(h1,h2),f1+f2))
10   return min(Q).first
```

⋆ Considerando que al menos `borraMin` tiene coste $O(\log n)$, el coste del algoritmo de Huffman está en $O(n \log n)$, siendo $n$ el número de caracteres distintos del fichero.

## 3. Demostración de optimalidad

⋆ **Lema 1:** Sean $x, y \in C$ los dos caracteres con menor frecuencia en el fichero. Entonces existe un árbol de Huffman óptimo $T$ en el que $x, y$ son dos hojas hermanas situadas a profundidad máxima en $T$.

*Demostración:* Sea $T$ un árbol óptimo que no cumple lo afirmado por el lema. Supongamos sin pérdida de generalidad que $f_x \leq f_y$. Sean $a, b$ dos hojas hermanas situadas a la máxima profundidad en $T$ y supongamos que $f_a \leq f_b$. Entonces tenemos $f_x \leq f_a$ y $f_y \leq f_b$.

Creamos $T' = T[x/a]$ y $T'' = T'[y/b]$, donde $T[c/d]$ denota el árbol que resulta de intercambiar en $T$ la hoja $c$ por la hoja $d$. Si empleamos $T'$ en lugar de $T$ para comprimir el fichero, la diferencia de longitudes del fichero comprimido es la siguiente:

$$\begin{aligned} L^T - L^{T'} &= \textstyle\sum_{c\in C} f_c.l_c^T - \sum_{c\in C} f_c.l_c^{T'} \\ &= f_x.l_x^T + f_a.l_a^T - f_x.l_x^{T'} - f_a.l_a^{T'} \\ &= f_x.l_x^T + f_a.l_a^T - f_x.l_a^T - f_a.l_x^T \\ &= \underbrace{(f_a - f_x)}_{\geq 0}\underbrace{(l_a^T - l_x^T)}_{\geq 0} \\ &\geq 0 \end{aligned}$$

Concluimos que $T'$ también es óptimo. Por un razonamiento similar demostraríamos que $L^{T'} - L^{T''} \geq 0$ y por tanto $T''$, en el que $x, y$ aparecen como hojas hermanas a la máxima profundidad, sería óptimo.

⋆ **Lema 2:** Sean $x, y \in C$ los dos caracteres con menor frecuencia en el fichero. Sea $C' = (C - \{x, y\}) \cup \{z\}$, donde $z$ es un carácter fresco al que asignamos la frecuencia $f_z = f_x + f_y$, es decir consideramos toda aparición de $x$ o $y$ en el fichero como una aparición de $z$. Sea $T'$ un árbol óptimo para el fihero con caracteres en $C'$. Entonces, el árbol $T$ obtenido reemplazando la hoja $z$ por el árbol $h(x) \bullet h(y)$, es óptimo para el fichero con caracteres en $C$.

*Demostración:* Por la forma en que ha sido construido $T$, deducimos:

1. $\forall c \in C - \{x, y\} \,.\, l_c^T = l_c^{T'}$
2. $l_x^T = l_y^T = l_z^{T'} + 1$
3. $f_x.l_x^T + f_y.l_y^T = (f_x + f_y)(l_z^{T'} + 1) = f_z.l_z^{T'} + f_x + f_y$

Entonces, la longitud del fichero comprimido con $T$ puede expresarse:

$$L^T = \sum_{c\in C} f_c.l_c^T = L^{T'} + f_x + f_y \quad \text{o bien} \quad L^{T'} = L^T - f_x - f_y \tag{1}$$

Supongamos ahora que $T$ no fuera óptimo para $C$. Entonces, existiría $T''$ óptimo tal que $L^{T''} < L^T$. Por el Lema 1, podemos suponer que en $T''$ aparecen $x, y$ como hojas hermanas situadas a la máxima profundidad. Construimos $T'''$ reemplazando en $T''$ el árbol $h(x) \bullet h(y)$ por la hoja $z$. $T'''$ sería entonces un árbol de Huffman para $C'$. Tendríamos entonces las siguientes inecuaciones:

$$\begin{aligned} L^{T'''} &= L^{T''} - f_x - f_y && \text{-- por construcción de } T''' \text{ y por (1)} \\ &< L^T - f_x - f_y && \text{-- por la hipótesis } L^{T''} < L^T \\ &= L^{T'} && \text{-- por (1)} \end{aligned}$$

Lo que contradice la hipótesis de que $T'$ es óptimo para $C'$. Concluimos entonces que $T$ es óptimo para $C$.

⋆ La corrección del algoritmo de Huffman se obtiene a partir del Lema 2 por inducción sobre el número $n$ de caracteres de $C$:

- $n = 1$ El árbol $T$ es una hoja con frecuencia 100 %. Obviamente $T$ es óptimo para $C$.

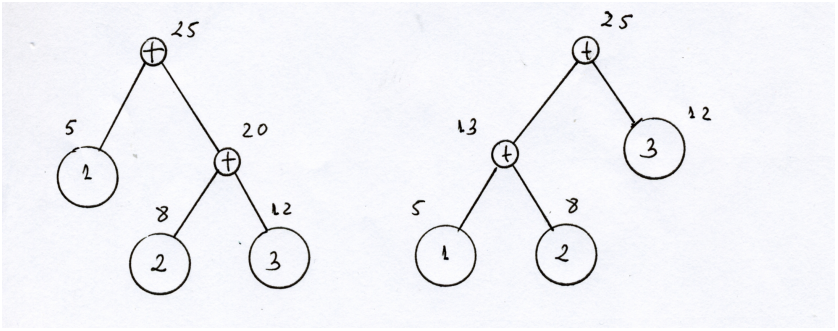
- $n > 1$ Eliminamos los dos caracteres $x, y$ con menor frecuencia y formamos un árbol con ellos. Consideramos que esos dos caracteres son indistinguibles, ya que insertamos el nuevo árbol como si fuera un caracter ficticio $z$ con una frecuencia suma de las de $x$ e $y$. Resolvemos el problema para $C - \{x, y\} \cup \{z\}$ que tiene $n - 1$ caracteres. Por hipótesis de inducción el árbol resultante es óptimo. En él, reemplazamos el carácter suma por el árbol en el que los dos caracteres están distinguidos. Por el Lema 2, también será óptimo.

## 4. Orden óptimo de mezcla de ficheros ordenados

⋆ Supongamos $n$ ficheros ordenados por orden creciente (por ejemplo, cada uno representa el conjunto de fichas de un grupo de alumnos). Sea $l_i$ la longitud del fichero $i, 1 \leq i \leq n$. Los ficheros han de mezclarse de dos en dos para producir un fichero total ordenado (en el ejemplo, se trataría de obtener un solo fichero con las fichas de todos los alumnos). Sabemos que el algoritmo de mezcla de dos ficheros de longitudes $l_1, l_2$ tiene un coste proporcional a $l_1 + l_2$. Se trata de elegir el orden de mezcla para que el coste total del conjunto de mezclas sea mínimo.

⋆ *Ejemplo:* Sean tres ficheros con $l_1 = 5,\ l_2 = 8,\ l_3 = 12$.

- El orden $1 \oplus (2 \oplus 3)$ tiene coste $l_1 + 2l_2 + 2l_3 = 45$
- El orden $(1 \oplus 2) \oplus 3$ tiene coste $2l_1 + 2l_2 + l_3 = 38$




⋆ El orden a seguir se puede representar mediante un árbol de Huffman, donde el número de veces que la longitud de un fichero interviene en la suma coincide con su profundidad en el árbol. Sea $p_i$ la profundidad de la hoja que representa el fichero $i$. El coste total se obtiene mediante la siguiente suma:

$$S = \sum_{i=1}^{n} p_i . l_i$$

⋆ El problema consiste entonces en encontrar el árbol de Huffman que minimiza $S$. Las $l_i$ son conocidas y las $p_i$ son las incógnitas. El problema es isomorfo pues al de la codificación óptima. Podemos emplear el algorimo voraz visto en la sección precedente para calcular dicho árbol.

⋆ El coste de calcular el órden óptimo de mezcla de $n$ ficheros ordenados está por tanto en $O(n \log n)$.

## Lecturas complementarias

Estas notas están basadas en la Sección 16.3 de [1], y en la Sección 4.7 de [2], donde el lector puede completar lo resumido aquí.

## Referencias

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 2nd edition, 2001.

[2] E. Horowitz, S. Sahni, and S. Rajasekaran. *Computer Algorithms*. Computer Science Press, 1998.