INF221 – Algoritmos y Complejidad

Clase #9
Código Huffman

Aldo Berrios Valenzuela

Martes 30 de agosto de 2016

1. Código Huffman

El código Huffman es una aplicación muy importante de algoritmo voraz.
Ámbito: compresión de datos.
Dado: Un texto, formado por caracteres. Buscamos codificarlo eficientemente. Cada caracter se codifica en k bits (total 2^k caracteres posibles), de texto de largo n usa nk bits.

En texto, las frecuencias son *muy* desiguales. Moby Dick: 117 194 veces 'c', 640 veces 'z'. Nuestro principal objetivo es asignarle codificaciones más largas a los caracteres que menos se repiten y codificaciones más cortas a aquellos que se repiten más.
Cuidado:

$$a \mapsto 0$$
$$b \mapsto 1$$
$$c \mapsto 01$$

Bajo esta codificación podemos escribir:

$$ababc \rightsquigarrow 010101 \tag{1.1}$$

Pero también:

$$ccc \rightsquigarrow 010101 \tag{1.2}$$

¡Se produce ambigüedad entre (1.1) y (1.2)!

Condición suficiente para evitarlo: Ningún código es prefijo de otro. Importante porque hace eficiente el decodificar ("prefix-free code" o "prefix code").

1.0.1. Descripción del problema

Dada una secuencia T sobre $\Sigma = \{x_1, \dots, x_n\}$, donde x_i aparece con frecuencia f_i, construir una función de codificación $C : \Sigma \to$ cadenas de bits, tal que C es un código prefijo y el número total de bits para representar T se minimiza.
Idea: Representar el código como árbol binario: cada arco se rotula con un bit 0 o 1 (digamos si va la izquierda lo rotulamos con 0, y si va a la derecha lo marcamos con 1)

Cada carácter rotula una de las hojas, el camino desde la raíz es el código de ese carácter (Figura 1).

- Es un código prefijo (llegó a la hoja x_i, ya no sigue, camino desde la raíz a la hoja es único).
- En el código óptimo, todo nodo interno tiene dos hijos (podemos "saltarnos" un nodo interno con un único hijo para crear un código más compacto).

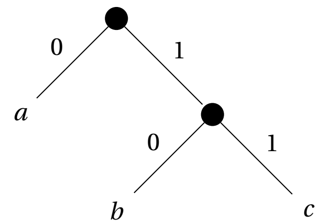

Figura 1: Los nodos que se encuentran en nivel más alto son los que más se repiten (mayor frecuencia) y las hojas de mayor profundidad son los caracteres que menos se repiten

Definición 1.1 (profundidad)**.** La *profundidad* de la hoja ℓ_i anotada $d(\ell_i)$, es el largo del camino de la raíz a esa hoja. El caracter x_i queda codificado por $d(x_i)$ bits, el texto completo por:

$$\sum_i f_i d(x_i) \quad \texttt{bits}$$

Intuitivamente, buscamos letras poco frecuentes a altas profundidades, frecuentes a profundidades bajas. Para ello, armamos el árbol desde las hojas. Vamos uniendo subárboles hasta tener uno solo.

Sea $L = (\ell_1, \dots, \ell_n)$ el conjunto de hojas para todos los caracteres, y sea f_i la frecuencia de la letra x_i. Hallar las dos letras de frecuencia mínima, digamos x_a y x_b con frecuencias f_a y f_b. Unir sus hojas en la hoja ℓ_{ab} con frecuencia $f_a + f_b$ dando un árbol R_{ab} (Figura 2):

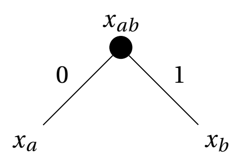

Figura 2: El nodo x_{ab} representa la unión entre los nodos x_a y x_b.

Recursivamente resolver el problema con:

$$L = \{\ell_1, \dots, \ell_n\} \setminus \{\ell_a, \ell_b\} \cup \{\ell_{ab}\} \tag{1.3}$$

y frecuencias ajustadas $(\ell_{ab} \rightsquigarrow f_a + f_b)$

Ejemplo 1.1. Considere el Cuadro 1. El algoritmo de Huffman nos pide encontrar los símbolos de menor frecuencia

Símbolo	Frecuencia
a	9
b	4
c	2
d	15
e	3
f	17

Cuadro 1: Dada una determinada secuencia de palabras, se detectó que sólo aparecen los símbolos a, b, c, d, e, f con las frecuencias adjuntas.

y crear un sub-arbol con ellos. En el cuadro 1, se tiene que los símbolos c y e son los que poseen menor frecuencia. Luego, formamos un sub-arbol con ellos (Figura 3).

De inmediato, agregamos este “nodo conjunto” al Cuadro 1, cuya frecuencia es equivalente al peso del árbol de la Figura 3 (suma de las frecuencias de c y e). El resultado se logra apreciar en el Cuadro 2.

Repetimos el proceso, es decir, escogemos dos símbolos del Cuadro 2 que tienen menor frecuencia y creamos un nuevo sub-árbol. Estos símbolos son ce y b. Entonces, la Figura 4 muestra el árbol resultante.

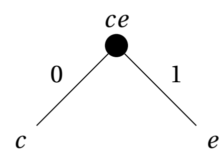

Figura 3: El presente árbol tiene como hojas los símbolos que menos se repiten.

Símbolo	Frecuencia
a	9
b	4
d	15
f	17
ce	5

Cuadro 2: Se agrega el nodo conjunto ce a la tabla, el cuál es considerado como un nuevo símbolo. La frecuencia de ce es la suma de las frecuencias de c y e.

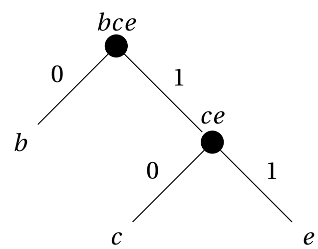

Figura 4: El presente árbol tiene como hojas los símbolos que menos se repiten.
Tiene un peso de $f_{ce} + f_b = 5 + 4 = 9$

Símbolo	Frecuencia
a	9
d	15
f	17
bce	9

Cuadro 3: Se reemplazan los símbolos b y ce del Cuadro 2 y se reemplaza con bce de frecuencia $f_{bce} = 9$

Inmediatamente, reemplazamos los símbolos b y ce del Cuadro 2 y lo reemplazamos con bce de frecuencia $f_{bce} = 9$. El resultado queda en el Cuadro 3.

Iteramos nuevamente. En el cuadro 3 se tiene que los dos símbolos con menor frecuencia son bce y a. Entonces, el árbol resultante queda representado por la Figura 5. Luego, quitamos estos símbolos del cuadro 3 y los reemplazamos por $abce$. El resultado se puede apreciar mirando el Cuadro 4.

Símbolo	Frecuencia
d	15
f	17
$abce$	18

Cuadro 4: Se reemplazan los símbolos bce y a del Cuadro 3 y se reemplaza con $abce$ que tiene una frecuencia $f_{abce} = f_{bce} + f_a = 18$

Continuamos iterando. Al observar el cuadro 4 vemos que los símbolos con menor frecuencia son d y f. Por lo tanto, tomamos estos dos símbolos y creamos un nuevo árbol que los tenga como hojas (Figura 6). En seguida, sacamos esos símbolos y lo reemplazamos por df. El resultado de hacer esto se puede observar en el cuadro 5.

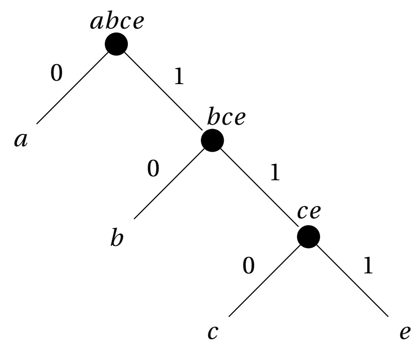

Figura 5: Este árbol tiene un peso de $f_{bce} + f_a = 18$.

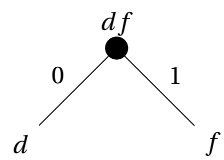

Figura 6: Las hojas de este árbol corresponden a los símbolos del cuadro 4 que tienen menor frecuencia. El peso es de $f_d + f_f = 32$

Símbolo	Frecuencia
df	32
$abce$	18

Cuadro 5: Se reemplazan los símbolos d y f del Cuadro 4 por df, que tiene una frecuencia de $f_{df} = f_d + f_f = 32$

Hacemos la última iteración, ya que sólo nos quedan dos símbolos. Tomamos los dos últimos símbolos y creamos el árbol final que se logra apreciar en la Figura 7.

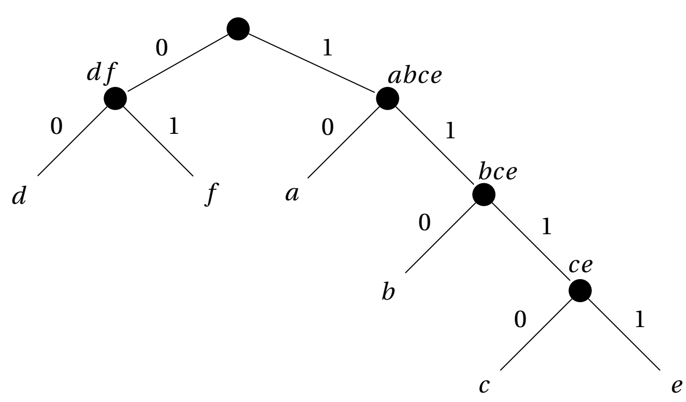

Figura 7: Este árbol tiene un peso de $f_{bce} + f_a = 18$.

■

Demostración (algoritmo voraz). Como siempre, para demostrar que este algoritmo es voraz, lo hacemos vía demostrar que:

- *Elección voraz:* Sea L la instancia original (o sea, el texto completo, con la frecuencia de cada símbolo respectivo), sean ℓ_a, y ℓ_b las hojas menos frecuentes. Entonces hay un árbol óptimo que incluye R_{ab}.

  Demostración. Sea R un árbol óptimo para L. Si R_{ab} es parte de R, salimos a carretear. Si el árbol R_{ab} *no* es parte de R, sean ℓ_x, ℓ_y dos hojas en R con padre común (hermanos), con $\delta = d(\ell_x) = d(\ell_y)$ máximo.

Entonces, suponiendo que ni a ni b son x ni y. Obtenga R^* intercambiando $x \leftrightarrow a$, $y \leftrightarrow b$, R^* contiene R_{ab}. Sea $B(R)$ el número de bits usados por el árbol R (importante, que en la demostración nos referiremos a d como el árbol original R).

Para el árbol R^*; con $d()$ referenciado del árbol original R:

$$\begin{aligned} B\left(R^*\right) &= B(R) - \left(f_x + f_y\right)\delta - f_a d(\ell_a) - f_b d(\ell_b) + \left(f_a + f_b\right)\delta + f_x d(\ell_a) + f_y d(\ell_b) \\ &= B(R) - \underbrace{\left(f_x - f_a\right)}_{\geq 0}\underbrace{(\delta + d(\ell_a))}_{\geq 0} - \underbrace{\left(f_y - f_b\right)}_{\geq 0}\underbrace{(\delta + d(\ell_b))}_{\geq 0} \end{aligned}$$

Pero R es óptimo. Por lo tanto, una contradicción. $\square$

$\square$