Código de Huffman, parte 1

Código de Huffman

Usado como mecanismo de compresión de varios protocolos y formatos. Uno de ellos es el MP3 en el que la señal se codifica en 3 pasos:

1. El audio análogo se digitaliza por muestreo en intervalos regulares en una secuencia de números reales s_1, s_2, …, s_T. Por ejemplo, a una tasa de 44100 muestras, una canción de 3 minutos corresponde a $T = 3 \times 60 \times 44100 \approx 8$ millones de valores
2. Cada valor real se aproxima a un número entero dentro de un conjunto finito A. Este conjunto se elige de tal forma que se exploten las limitaciones auditivas del ser humano.
3. El arreglo resultante de tamaño T sobre el alfabeto A se codifica a binario. En este último paso es donde entra en juego el código de Huffman

Código de Huffman

Se basa en códigos binarios y bien conocido por su efectividad: ahorros de hasta el 80% (dependiendo del tipo de archivo y contenido específico).

Un código binario mapea cada carácter de un alfabeto a un string binario. Un ejemplo típico es el código ASCII de 256 caracteres el cual emplea 8 bits.

En general, para un alfabeto de *n* caracteres la forma típica de codificación es el uso de un string de una longitud de *log(n)* bits.

¿Es esta la única alternativa?

Resulta que no. Si las frecuencias de aparición de los caracteres no es homogénea, es más eficiente usar códigos binarios de longitud variable.

Código de Huffman

Supongamos por ejemplo que un archivo contiene 100.000 caracteres pero solo entre ‘a’ y ‘f’ y que sus frecuencias de aparición (en miles) son:

a	b	c	d	e	f
45	13	12	16	9	5

Un código binario de longitud fija requeriría de 3 bits: a=000, b=001, c=010, d=011, e=100, y f = 101. El tamaño del archivo en este caso sería de 300.000 bits.

Un código binario de longitud variable aprovecha el conocimiento de la frecuencia de aparición de los caracteres para asignarle menos bits a los caracteres con mayor frecuencia y más bits para los de menor.

Código de Huffman

Volviendo al ejemplo, podríamos tener el siguiente código binario de longitud variable:

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

En este caso el tamaño del archivo sería:

(45*1 + 13*3 + 12*3 + 16*3 + 9*4 + 5*4)*1000 = 224.000 bits

Lo que representa un ahorro de cerca del 25%

La pregunta es entonces: ¿cómo definir ese mapa de códigos lo más eficientemente posible?