Algoritmo de Huffman

Inconvenientes que originan el uso del algoritmo :

1) Insuficiente cantidad de espacio en discos y diskettes.
2) Tiempos de transmisión prolongados y costos elevados.

Introducción .

Por lo general, las computadoras codifican los caracteres mediante códigos como el ASCII y el EBCDIC. En el caso del ASCII, se maneja con 8 bits para la representación de un byte, por lo tanto hay 2^8 posibles representaciones de caracteres pero siempre de longitud fija.
Una posibilidad de disminuir el espacio, sería la de utilizar códigos de representación de longitud variable.
Así por ejemplo, podría representarse la letra **a** con el valor binario **0**, la letra **b** con el valor binario **1**, la letra **c** con el valor **01** y asi sucesivamente. Pero ante la codificación **0101** podría interpretarse como **abab** o bien como **cc**.
El problema de decodificar un **01** podría solucionarse mediante la utilización de un prefijo antes de la codificación del caracter que indique el comienzo de un caracter y el fin del anterior, pero esto encarecería nuevamente el tema del espacio que es el que ahora nos preocupa.

Concepto del algoritmo.

El algoritmo de Huffman define un código variable para cada caracter sin la necesidad de utilizar un prefijo. De esta manera es posible que, aquellos caracteres que son más frecuentes dentro de un alfabeto, se codifiquen con la menor cantidad de bits posibles mientras que aquellos que no sean tan frecuentes podrán tener una codificación un poco más amplia, pero siempre tendiendo a ser la menor posible. Es obvio pensar que en el alfabeto del lenguaje castellano la letra **a** es usada con mucha más frecuencia que la letra **x**. Este concepto se optimizaría si en lugar de aplicarlo a un alfabeto se aplicara a cada texto en particular, es decir, conocer con exactitud cuantas veces se repite un caracter dentro del texto para, de esa manera darle un código variable que sea el menor posible para aquellos que se repitan con mayor frecuencia y a medida que sea necesario utilizar mas bits para codificar un caracter, que la frecuencia con que el caracter se usa en el texto sea menor.

Para ello, Huffman hace uso de una tabla de frecuencias donde se guardan los caracteres empleados en el texto y la cantidad de veces que son utilizados y de un árbol binario como estructura de datos. El código generado por el algoritmo, está basado en el código Morse y define una longitud variable de codificación basada en valores estadísticos.

Tabla de Frecuencias.

Huffman toma el archivo a compactar y genera una tabla de frecuencias con los caracteres utilizados. Por ejemplo supongamos que el archivo a compactar tiene el siguiente texto (no tomar en cuenta las comillas) :

"tomamos como ejemplo esta frase " .

La tabla de frecuencias sería la siguiente :

char	**frec**	**codigo**	**bits**	**Total**
blanco	5	011	3	15
o	5	010	3	15
m	4	000	3	12
e	4	001	3	12
a	3	110	3	9
s	3	111	3	9
t	2	1011	4	8
c	1	10100	5	5
j	1	10101	5	5
p	1	10011	5	5
l	1	10010	5	5
f	1	10001	5	5
r	1	10000	5	5
Total				110

Si se hubiera codificado mediante código ASCII, la cantidad de bits utilizados sería igual a 256, es decir que mediante el código de Huffman, sin ninguna optimización en especial, se obtuvo un archivo compactado en un 57%. ***Como obtiene Huffman el código de cada caracter...?*** Mediante el uso de :

a) una *tabla de frecuencias*
b) un *árbol binario* como estructura de datos donde :

* cada hijo izquierdo distinto de nil representa un 0 binario
* cada hijo derecho distinto de nil representa un 1 binario
* cada hoja del árbol es un caracter

En el ejemplo planteado, el árbol asociado sería el de la *Figura 1. :*

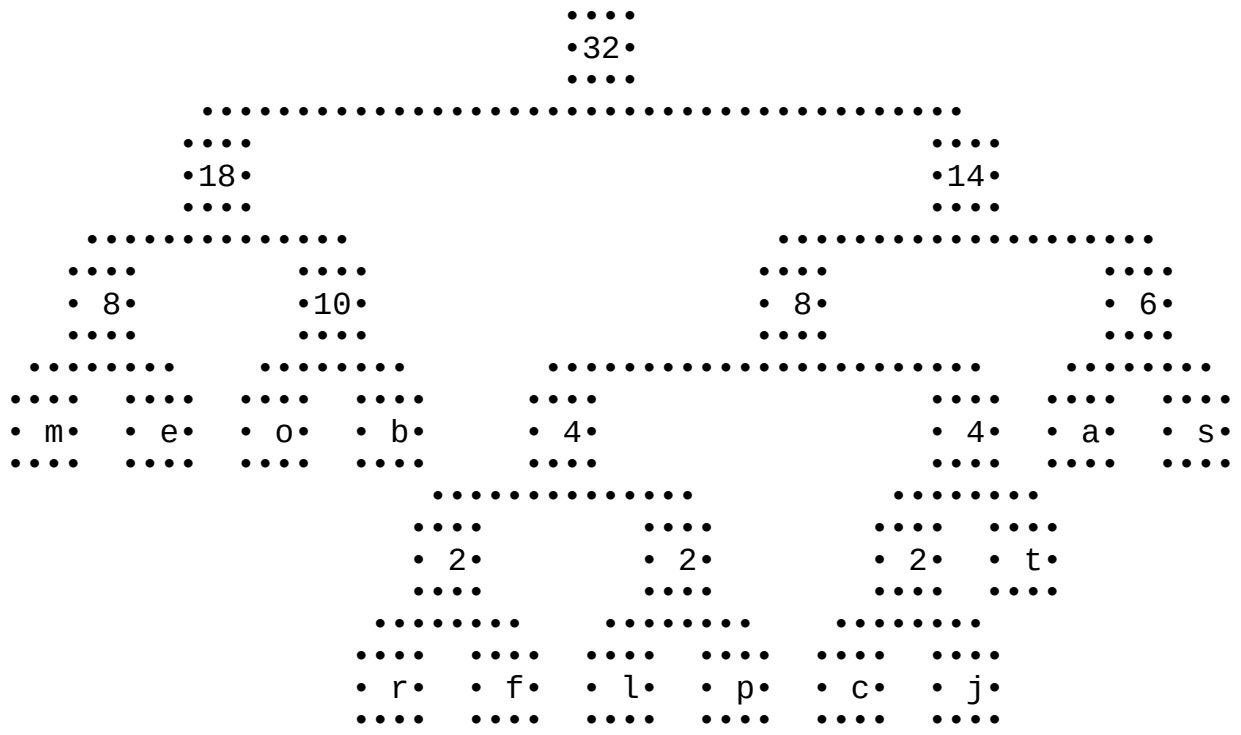

Figura 1

El nodo del árbol binario que utiliza Huffman tiene el siguiente diseño mostrado en la **Figura 2** *:*

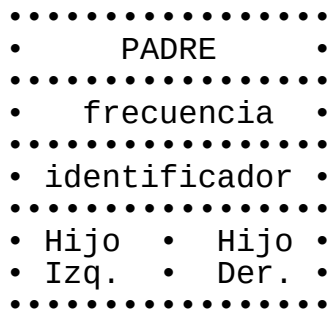

Figura 2

El algoritmo se basa en :

1) Hallar los dos caracteres con la frecuencia más baja de la tabla de frecuencias. Si un caracter tiene frecuencia cero se ignora. Si dos o más caracteres tienen igual frecuencia se eligen dos cualesquiera.

2) Combinar los dos caracteres en un arbol binario. (Ver *Figura 3*).
Cada hoja de este árbol contiene uno de los caracteres que se quiere codificar en el campo *identificador*, la frecuencia obtenida de la tabla de frecuencias en el campo *frecuencia*, nil en los campos *hijo izq.* e *hijo der.* y la dirección de la raiz del árbol en el campo padre.
La raíz del árbol (o el padre de las hojas) contiene blancos o nulos en el campo identificador, la sumatoria de las frecuencias de sus hijos en el campo *frecuencia* y la dirección de cada hijo en los campos *hijo der*. e *hijo izq.* respectivamente.

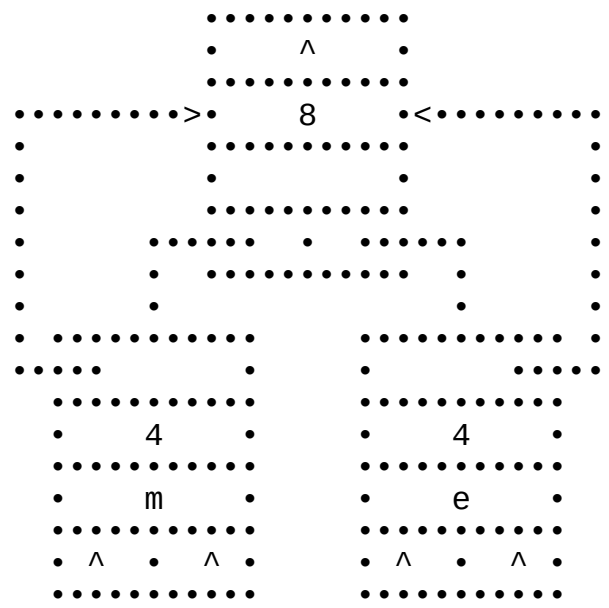

Figura 3

3) Ejecutar nuevamente el punto 2). Esta vez entra en la comparación el valor de la frecuencia del árbol recientemente creado en lugar de las frecuencias de sus caracteres
Este proceso se ejecuta hasta que quede sólo un árbol para todos los caracteres de la tabla de frecuencias y que de el pueda ser leído un char directamente.

Observaciones :

* La optimización del algoritmo implica ocupar el menor espacio posible, por lo cual hay que tratar de disminuir la profundidad de los nodos maximales, es decir, la longitud de paso máximo.

* El problema del prefijo, que se presenta en las codificaciones de longitud variable queda solucionado en el árbol de Huffman al ser el nodo maximal el que contiene el caracter. De esa manera, al detectar un nodo maximal, es decir **Hijo Der. = Hijo Izq. = nil** puedo asegurar que en ese momento de la lectura de bits se produce el fin de la codificación de un caracter y que con el siguiente bit comenzará el próximo caracter.

Codificación y Decodificación

Una vez que se obtuvo la tabla de frecuencias y de esta, el árbol binario estamos en condiciones de ***decodificar*** un tren de bits. Así por ejemplo si el tren de bits a decodificar es el siguiente :

000 010 111 10010
m o s l

Por cada bit leído se deberá recorrer el árbol binario comenzando por la raíz y visitando los subárboles izquierdo o derecho según corresponda, de acuerdo a si el bit leído es un **0** ó un **1** respectivamente.
Se deduce de la lectura del árbol que el string es "**mosl**"

Para ***codificar*** un caracter, será necesario guardar la dirección que le fue asignada al nodo al haber sido dado de alta en el árbol. Una solución sería la de agregar otra columna en la tabla de frecuencias que contenga las direcciones físicas que ocupan los nodos que contienen los caracteres.
Obtenida la dirección del nodo que contiene el caracter que se quiere codificar, se recorre el árbol en forma ascendente mediante la dirección guardada en el campo que apunta al padre del nodo. Al bajar de nivel, en dirección a la raíz del árbol, se recupera un **0** ó un **1** en función de si el hijo es izquierdo (se codifica un **0**) o si es derecho (un **1**). Cada bit obtenido, es ingresado en una pila. Este proceso se ejecuta hasta llegar a la raíz del árbol (campo **PADRE** = nil). En este momento se tiene la codificación del caracter invertida en la pila por lo que al vaciar la pila, se obtendrá la codificación binaria del caracter.