Análisis y Diseño de Algoritmos

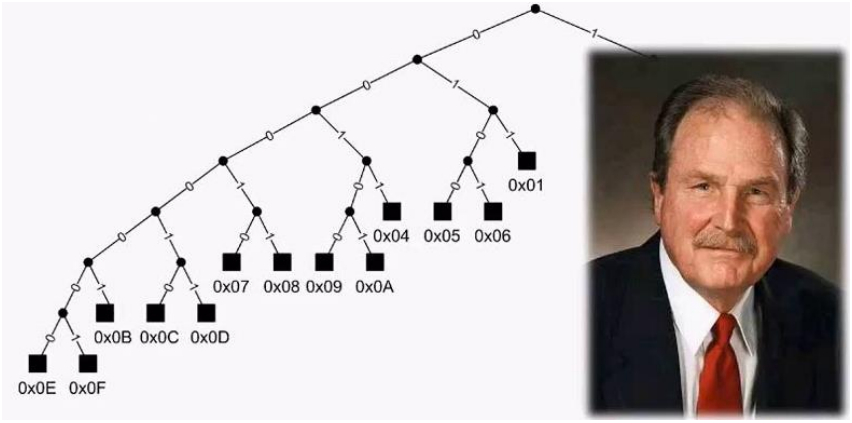
Lucia Iveth De La Vega Hernández 3CM2

Actividad: Códigos de Huffman

¿Qué son?

El algoritmo de Huffman provee un método que permite comprimir información mediante la recodificación de los bytes que la componen. En particular, si los bytes que se van a comprimir están almacenados en un archivo, al recodificarlos con secuencias de bits más cortas diremos que lo comprimimos.

La codificación Huffman (también conocida como Codificación Huffman) es un algoritmo para realizar la compresión de datos y forma la idea básica detrás de la compresión de archivos. Esta publicación habla sobre la codificación de longitud fija y de longitud variable, los códigos decodificables únicos, las reglas de prefijo y la construcción del árbol de Huffman.



¿Para qué sirven?

La técnica consiste en asignar a cada byte del archivo que vamos a comprimir un código binario compuesto por una cantidad de bits tan corta como sea posible. Esta cantidad será variable y dependerá de la probabilidad de ocurrencia del byte. Es decir: aquellos bytes que más veces aparecen serán recodificados con combinaciones de bits más cortas, de menos de 8 bits. En cambio, se utilizarán combinaciones de bits más extensas para recodificar los bytes que menos veces se repiten dentro del archivo. Estas combinaciones podrían, incluso, tener más de 8 bits.

Los códigos binarios que utilizaremos para reemplazar a cada byte del archivo original se llaman “códigos Huffman”.

¿Cómo se aplican los árboles binarios?

Se llevan una serie de pasos:

1. **Contar la cantidad de ocurrencias de cada carácter**
   El primer paso consiste en contar cuántas veces aparece en el archivo cada carácter o byte. Como un byte es un conjunto de 8 bits, resulta que solo existen 28 = 256 bytes diferentes.

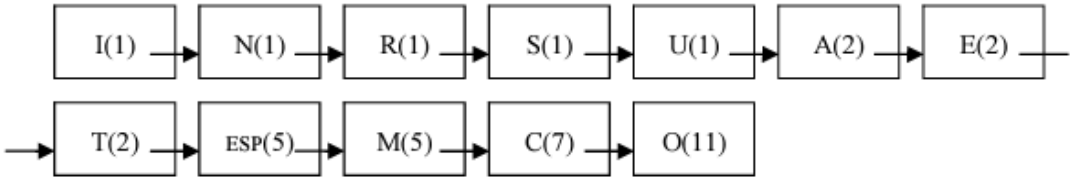
Entonces utilizaremos una tabla con 256 registros para representar a cada uno de los 256 bytes y sus correspondientes contadores de ocurrencias.

	Carácter	*n*
0		
:		
32	*ESP*	5
:		
65	A	2
:		
67	C	7
:		
69	E	2
:		
73	I	1

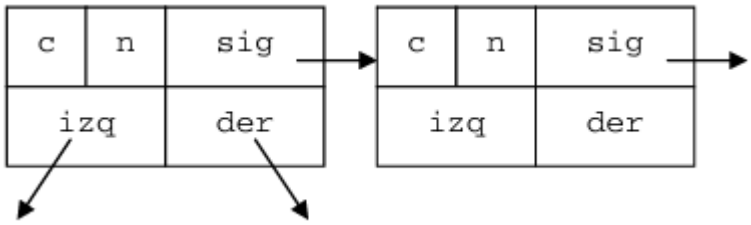
:	Carácter	*n*
:		
77	M	5
78	N	1
79	O	11
:		
82	R	1
83	S	1
84	T	2
85	U	1
:		
255		

2. Crear una lista enlazada

Conociendo la cantidad de ocurrencias de cada carácter, tenemos que crear una lista enlazada y ordenada ascendentemente por dicha cantidad. Primero los caracteres menos frecuentes y luego los que tienen mayor probabilidad de aparecer y, si dos caracteres ocurren igual cantidad de veces, entonces colocaremos primero al que tenga menor valor numérico. Por ejemplo: los caracteres 'I', 'N', 'R', 'S' y 'U' aparecen una sola vez y tienen la misma probabilidad de ocurrencia entre sí; por lo tanto, en la lista ordenada que veremos a continuación los colocaremos ascendentemente según su valor numérico o código ASCII.



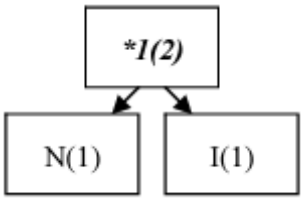
Los nodos de la lista tendrán la siguiente estructura:



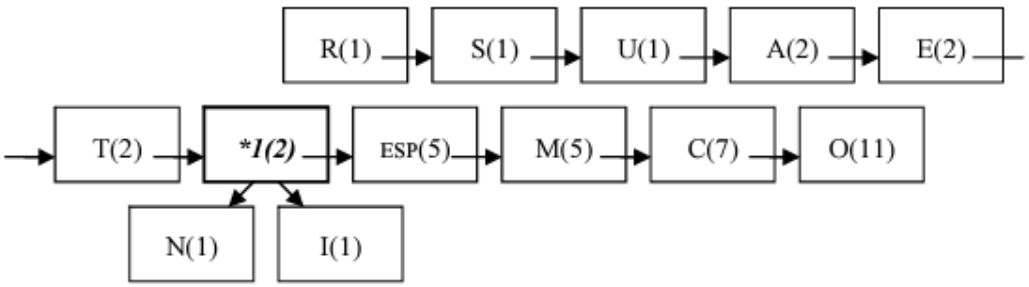
El campo c representa el carácter (o byte) y el campo n la cantidad de veces que aparece dentro del archivo. El campo sig es la referencia al siguiente elemento de la lista enlazada. Los campos izq y der, más adelante, nos permitirán implementar el árbol binario. Por el momento no les daremos importancia; simplemente pensemos que son punteros a null.

3. **Convertir la lista enlazada en el árbol Huffman**

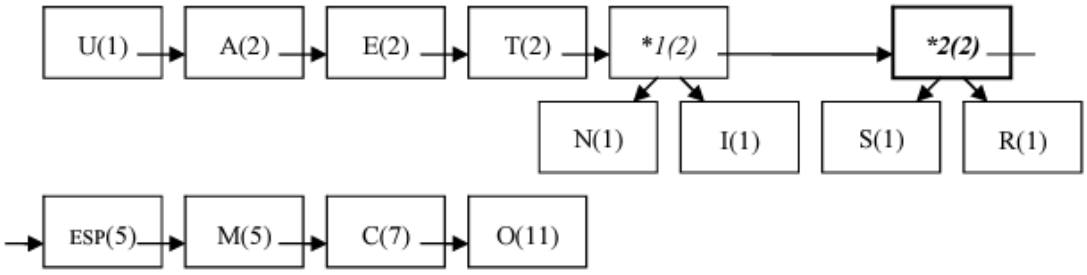
Vamos a generar el árbol Huffman tomando "de a pares" los nodos de la lista. Esto lo haremos de la siguiente manera: sacamos los dos primeros nodos y los utilizamos para crear un pequeño árbol binario cuya raíz será un nuevo nodo que identificaremos con un carácter ficticio y una cantidad de ocurrencias igual a la suma de las cantidades de los dos nodos que estamos procesando. En la rama derecha colocamos al nodo menos ocurrente (el primero); el otro nodo lo colocaremos en la rama izquierda.



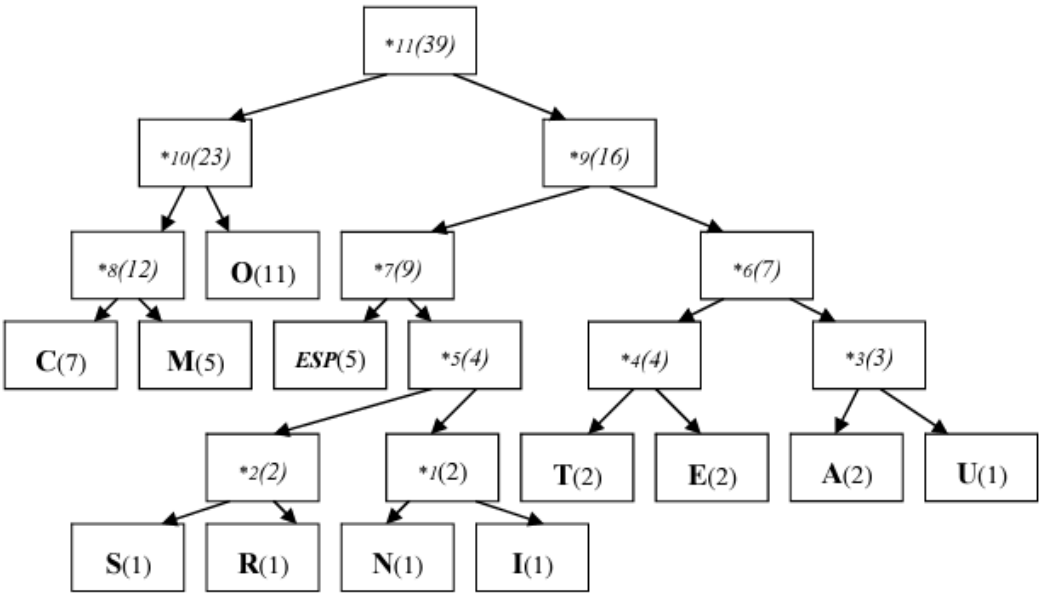
Luego insertamos en la lista al nuevo nodo (raíz) respetando el criterio de ordenamiento que mencionamos más arriba. Si en la lista existe un nodo con la misma cantidad de ocurrencias (que en este caso es 2), la inserción la haremos a continuación de este.



Ahora repetimos la operación procesando nuevamente los dos primeros nodos de la lista: R(1) y S(1):



Luego continuamos con este proceso hasta que la lista se haya convertido en un árbol binario cuyo nodo raíz tenga una cantidad de ocurrencias igual al tamaño del archivo que queremos comprimir.



Observemos que el árbol resultante tiene caracteres ficticios en los vértices y caracteres reales en las hojas.

4. Asignación de códigos de Huffman

El siguiente paso será asignar un código Huffman a cada uno de los caracteres reales que se encuentran ubicados en las hojas del árbol. Para esto, consideraremos el camino que se debe recorrer para llegar a cada hoja. El código se forma concatenando un 0 (cero) por cada tramo que avanzamos hacia la izquierda y un 1 (uno) cada vez que avanzamos hacia la derecha. Por lo tanto, el código Huffman que le corresponde al carácter 'O' es 01, el código que le corresponde al carácter 'M' es 001 y el código que le corresponde al carácter 'S' es 10100. Como podemos ver, la longitud del código que el árbol le asigna a cada carácter es inversamente proporcional a su cantidad de ocurrencias. Los caracteres más frecuentes reciben códigos más cortos mientras que los menos frecuentes reciben códigos más largos. Agreguemos los códigos Huffman (cod) y sus longitudes (nCod) en la tabla de ocurrencias.

	Carácter	n	cod	nCod
0				
:				
32	ESP	5	100	3
:				
65	A	2	1110	4
:				
67	C	7	000	3
:				
69	E	2	1101	4
:				
73	I	1	10111	5

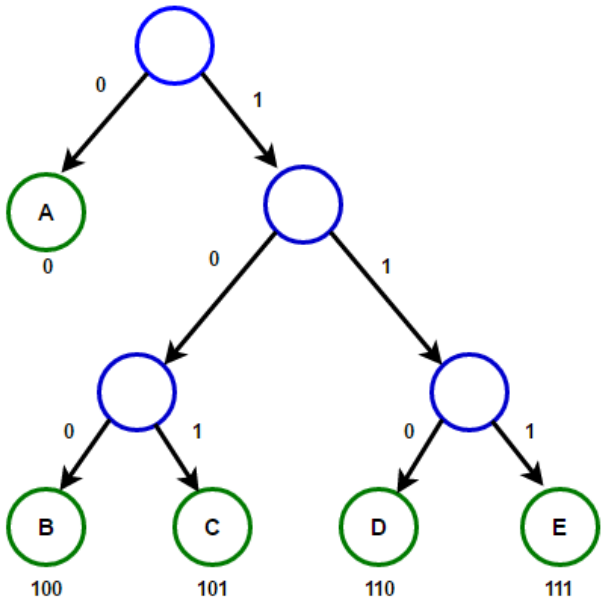
:	Carácter	n	cod	nCod
:				
77	M	5	001	3
78	N	1	10110	5
79	O	11	01	2
:				
82	R	1	10101	5
83	S	1	10100	5
84	T	2	1100	4
85	U	1	11111	5
:				
255				

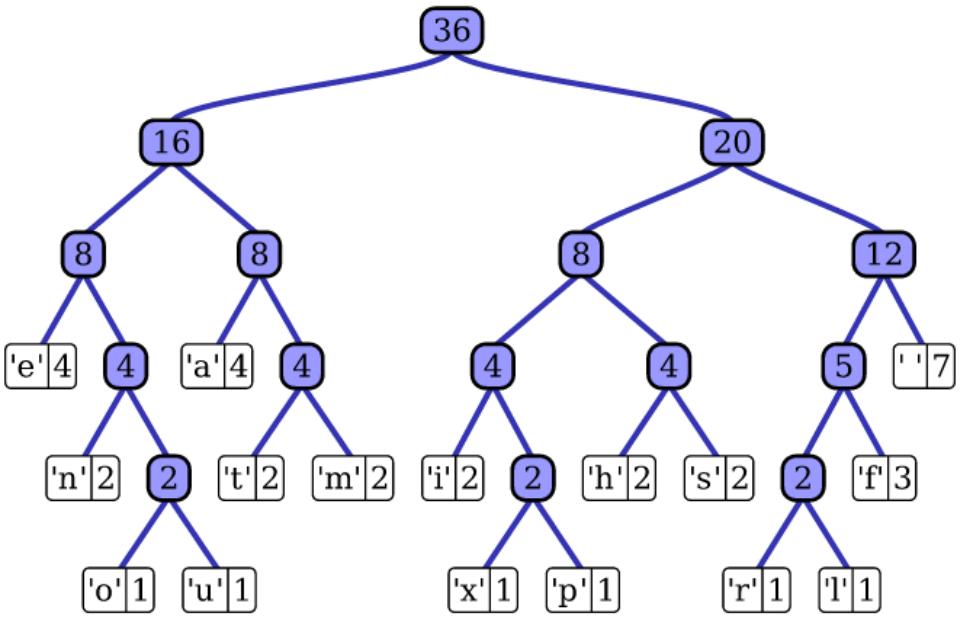
5. **Codificación del contenido**

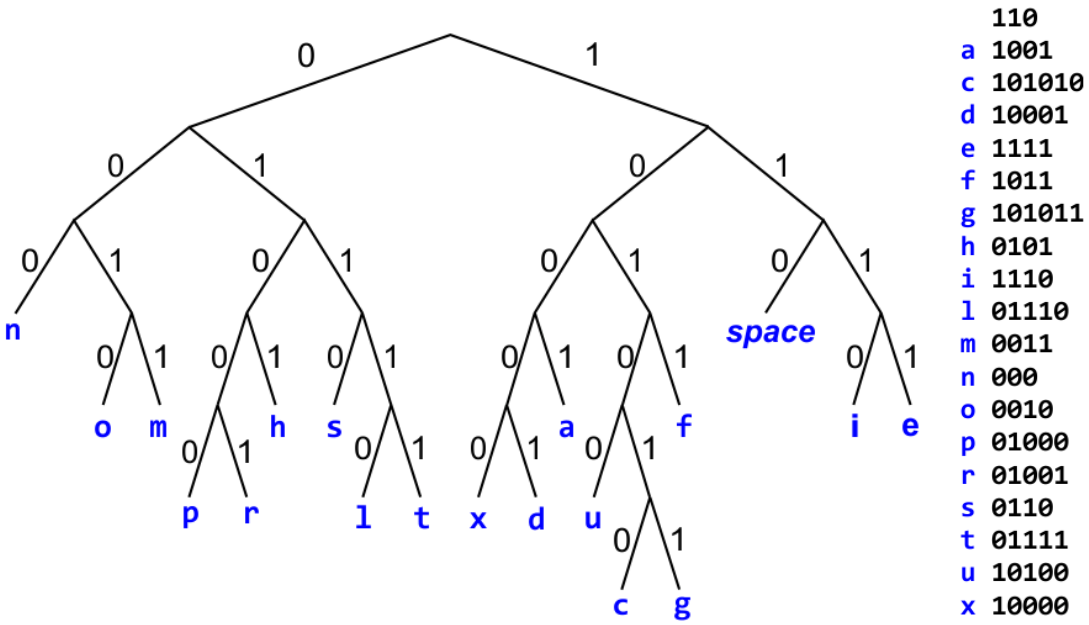
Para finalizar, generamos el archivo comprimido reemplazando cada carácter del archivo original por su correspondiente código Huffman, agrupando los diferentes códigos en paquetes de 8 bits ya que esta es la menor cantidad de información que podemos manipular.

C	O	M	O	*[esp]*	C	...
000	01	001	01	100	000	...
00001001			01100000			...

3 ejemplos manuales








```
   110
a 1001
c 101010
d 10001
e 1111
f 1011
g 101011
h 0101
i 1110
l 01110
m 0011
n 000
o 0010
p 01000
r 01001
s 0110
t 01111
u 10100
x 10000
```

Referencias

- https://libroweb.alfaomega.com.mx/book/393/free/data/Capitulos/cap16.pdf
- https://www.techiedelight.com/es/huffman-coding/
- http://sintesis.ugto.mx/WintemplaWeb/02Wintempla/37Libraries/05Huffman%20coding/index.htm
- https://ramon-gzz.blogspot.com/2013/04/codificacion-huffman.html