

COMPRESIÓN DE ARCHIVOS

Diseño y Análisis de Algoritmos
Cátedra de Programación
Carrera de Ingeniería de Sistemas



UNIVERSIDAD
DE LOS ANDES
MERIDA VENEZUELA

Gabriel Omaña
C.I: 15.175.053

COMPRESIÓN DE ARCHIVOS



Los algoritmos que se han estudiado hasta ahora han sido diseñados, en su mayor parte para que utilicen el menor tiempo posible, quedando la economía de espacio en un segundo plano.

La compresión de archivos concibe la creación de algoritmos de forma inversa, es decir, utilizar el menor espacio posible dejando a un lado el tiempo

COMPRESIÓN DE ARCHIVOS



➤ Codificación por Longitud de series

El tipo mas simple de redundancia que se puede encontrar en un archivo son las largas series de caracteres repetidos.

"AAAABBBBAABBBBBBCCCCCCCCDABCBAABBBBCCCD"

Esta cadena se puede codificar de forma mas compacta remplazando cada repetición de caracteres por un solo ejemplar del carácter repetido precedido del numero de veces que este se repite es decir:

"4A33BAA5B8CDABC3A4B3CD"

COMPRESIÓN DE ARCHIVOS



➤ Codificación por Longitud de series

El tipo mas simple de redundancia que se puede encontrar en un archivo son las largas series de caracteres repetidos.

"AAAABBBABBBBBBCCCCCCCCDABCBAAABBBBBCCCD"

Esta cadena se puede codificar de forma mas compacta remplazando cada repetición de caracteres por un solo ejemplar del carácter repetido precedido del numero de veces que este se repite, es decir:

"4A3BAA5B8CDABCB3A4B3CD"

COMPRESIÓN DE ARCHIVOS



Si se utilizan otros caracteres para codificar las longitudes de las series, no podría aplicarse el método a las cadenas que contengan esos caracteres.

Para solucionar ese problema se aplica lo siguiente:

1. Se escoge el carácter con la menor probabilidad de aparecer (*carácter de escape*)
2. Cada aparición del *carácter de escape*, esta seguida de un carácter cuya posición en el alfabeto representa el numero de repeticiones del carácter que se esta codificando
3. Luego vendrá el carácter que se repite

De esta manera se construye la llamada *secuencia de escape*, formada por, *<carácter de escape, longitud, carácter repetido>*



COMPRESIÓN DE ARCHIVOS

EJEMPLO:

Sea un alfabeto integrado por 26 letras, con el cual se construyo la cadena "AAABBBBAABBBBBBCCCCCCCCDABCBAABBBBBCCCD", Y se escoge como *carácter de escape* a 'Q', aplicando el método obtendremos:

"QDABBBAAQEBOHC DABCBA AAAQDBCCCD"

Secuencia de Escape { Q: *Carácter de Escape*
D: *Carácter de longitud (4)*
A: *Carácter Repetido*

AAA → 4A

COMPRESIÓN DE ARCHIVOS



➤ Codificación de Longitud Variable

Esta técnica de compresión permite ganar una cantidad considerable de espacio en archivos, la idea es utilizar pocos bits para los caracteres que aparecen habitualmente y unos pocos mas para los que aparecen menos.

Tomemos como ejemplo la cadena "ABRACADABRA", cuya representación en código binario esta dada por 5 bits de i que reproducen la i -esima letra del alfabeto (0 para los blancos)

00001	00010	10010	00001	00011	00001	00100	00001	00010	10010	00001
A	B	R	A	C	A	D	A	B	R	A

Nótese que 'D' aparece una vez en la cadena mientras que 'A' aparece cinco, sin embargo ambos caracteres necesitan el mismo numero de bits



COMPRESIÓN DE ARCHIVOS

Para aplicar el método se asigna de forma proporcional la cadena mas corta de bits a la letra mas frecuente y así sucesivamente con todos los caracteres, para nuestro caso:

A → 0 , B → 1 , R → 01 , C → 10 , D → 11

0	1	01	0	10	0	11	0	1	01	0
A	B	R	A	C	A	D	A	B	R	A

Esta cadena utiliza solo 15 bits en lugar de los 55 que usaba la anterior. Aunque es cierto que depende de los espacios en blancos pues sin ellos se podría decodificar de forma incorrecta, sin embargo agregar 10 bits mas en delimitadores no hace que sea mas grande que la original



COMPRESIÓN DE ARCHIVOS

Por lo que podríamos volver a codificar la cadena "ABRACADABRA" de la siguiente manera

A → 11 , B → 00 , R → 011 , C → 010 , D → 10

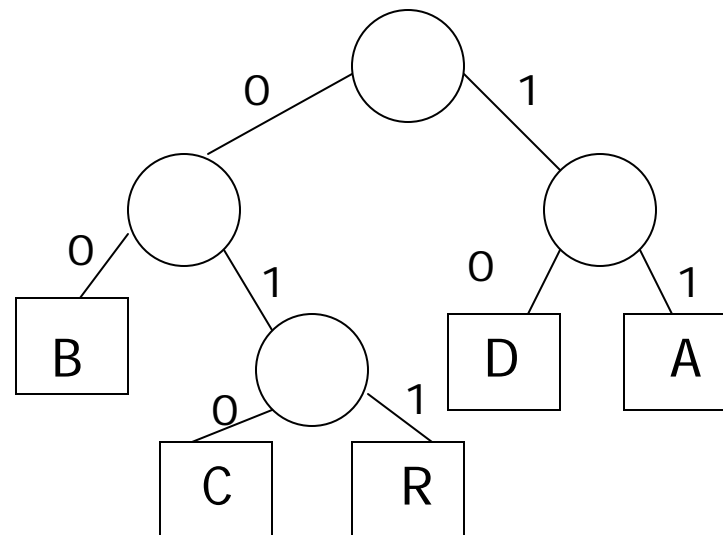
11	00	011	11	010	11	10	11	00	011	11
A	B	R	A	C	A	D	A	B	R	A

Esta cadena utiliza 25 bits en lugar de 55, por lo que sigue siendo mas pequeña que la original y posee una única forma de decodificar

COMPRESIÓN DE ARCHIVOS

Una forma fácil de representar el código es mediante un árbol. En efecto cualquier árbol de M hojas es capaz de representar un mensaje de M caracteres diferentes, para nuestro caso:

11	00	011	11	010	11	10	11	00	011	11
A	B	R	A	C	A	D	A	B	R	A



COMPRESIÓN DE ARCHIVOS



La representación por un árbol garantiza que el código es unívocamente decodificable a partir del árbol.

Pero el problema radica en que se pueden obtener distintos árboles para un mismo mensaje dependiendo de cómo haya sido codificado, entonces cual es el mejor?

Existe una forma elegante de construir un árbol que proporcione una cadena de bits de longitud mínima para cualquier mensaje. Este método fue creado por *D. Huffman* en 1952, y se conoce como Algoritmo de Huffman

COMPRESIÓN DE ARCHIVOS



➤ Código de Huffman

Los códigos de Huffman son una técnica muy útil para comprimir ficheros, este utiliza una tabla de frecuencias de aparición de cada carácter para construir una forma óptima de representar los caracteres con códigos binarios.

Los pasos en la construcción del código de huffman son:

1. Se cuenta el numero de veces que se repite un carácter (frecuencia de aparición), en el mensaje que se va a codificar
2. Se construye el árbol de abajo hacia arriba de acuerdo con las frecuencias. Al construir el árbol se considera como un árbol binario, luego como un árbol de codificación

COMPRESIÓN DE ARCHIVOS



Los pasos en la construcción del árbol son:

1. Se crea un nodo de árbol para cada frecuencia distinta de cero, asociada a su carácter
2. Se escogen los nodos con las frecuencias mas pequeñas y se unen en un nodo cuyos hijos serán los que uní y la frecuencia del nodo nuevo será la suma da las frecuencias de los nodos unidos
3. Continuar este proceso hasta que se forme un árbol único

NOTA: En caso de que en una serie de nodos hayan mas de 2 nodos que cumplen con ser los mas pequeños, se escoge arbitrariamente los que se unirán. Esta escogencia genera árboles distintos pero equivalentes desde el punto de vista de la optimidad

COMPRESIÓN DE ARCHIVOS



EJEMPLO:

Se tiene un fichero con 100.000 caracteres que se desea compactar. Las frecuencias de aparición de caracteres en el fichero son las siguientes:

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

Codificando este fichero mediante el uso del código de longitud variable, y asignando la menor cadena de bits al carácter que mas se repite, obtenemos lo siguiente:

	a	b	c	d	e	f
cód.long.var.	0	101	100	111	1101	1100

COMPRESIÓN DE ARCHIVOS

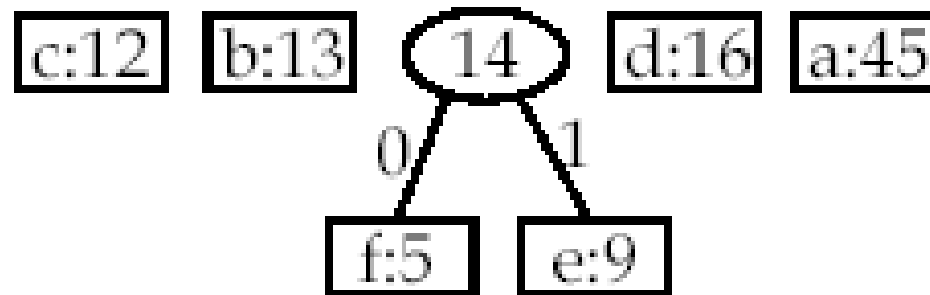


Construyendo el árbol.

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

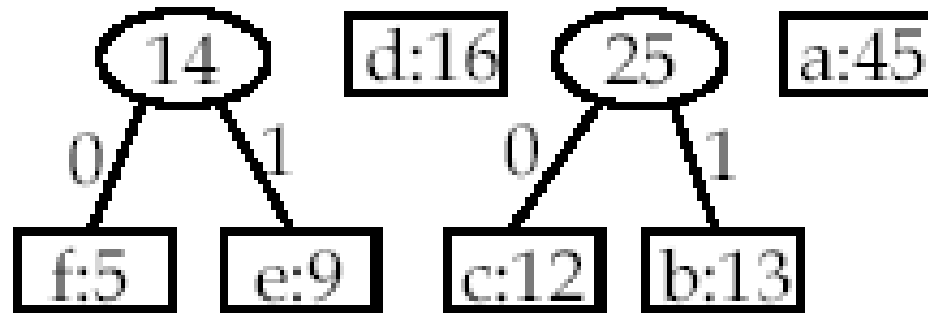
COMPRESIÓN DE ARCHIVOS

Construyendo el árbol..



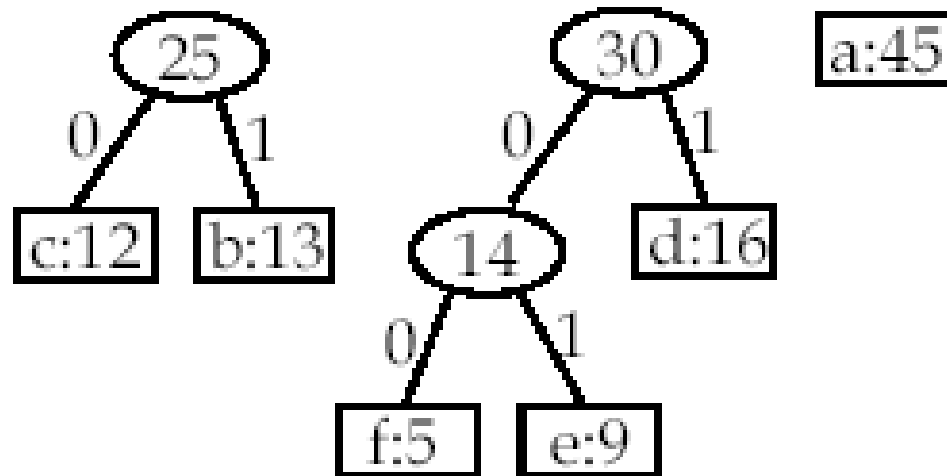
COMPRESIÓN DE ARCHIVOS

Construyendo el árbol...



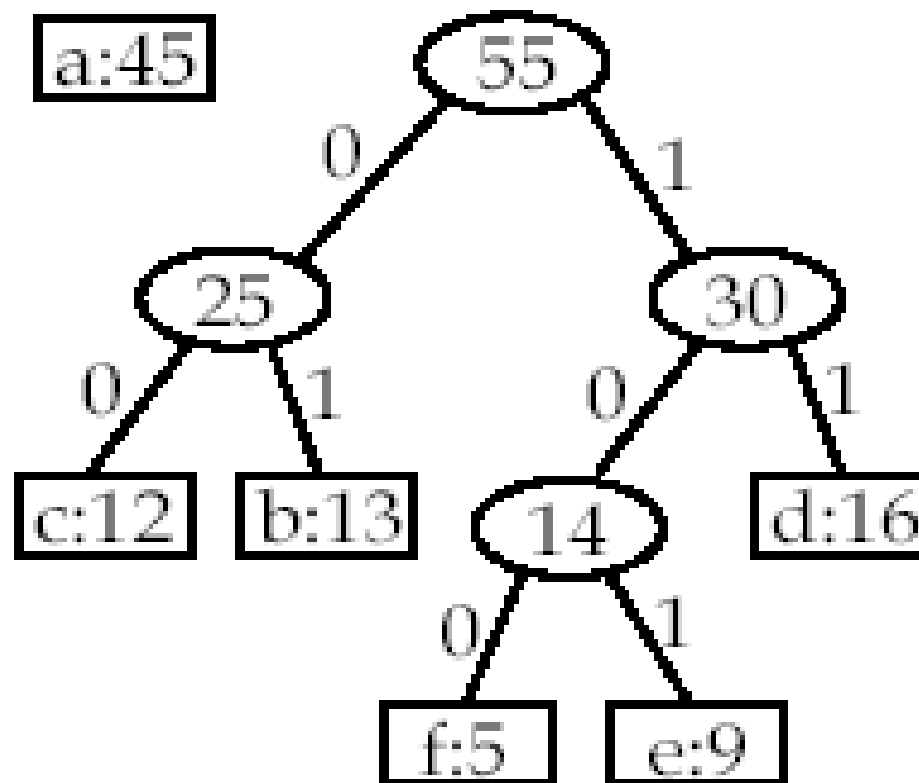
COMPRESIÓN DE ARCHIVOS

Construyendo el árbol....



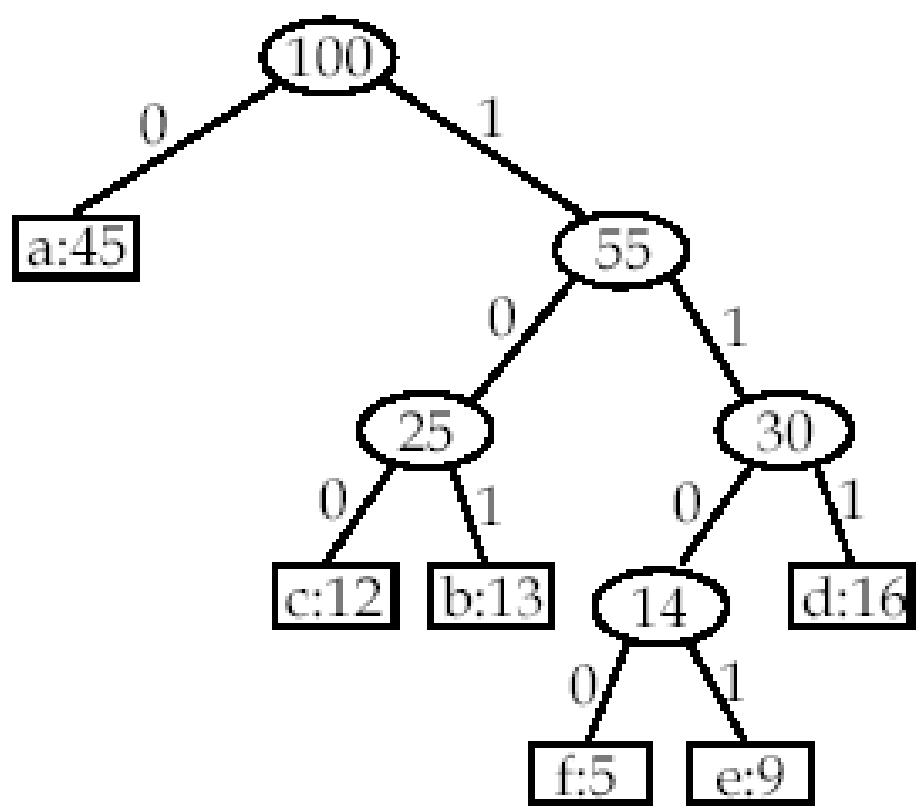
COMPRESIÓN DE ARCHIVOS

Construyendo el árbol.....



COMPRESIÓN DE ARCHIVOS

Finalmente....



COMPRESIÓN DE ARCHIVOS



Evidentemente las letras con frecuencias altas están cerca de la raíz del árbol y se codifican con pocos bits, por lo que genera un buen código.

Propiedades de los árboles del código huffman:

1. La longitud del mensaje codificado es igual a la longitud ponderada del camino externo del árbol de frecuencias de huffman.
2. Ningún árbol con la misma frecuencia en los nodos externos puede tener una longitud ponderada del camino externo inferior a la del árbol de huffman

COMPRESIÓN DE ARCHIVOS

Huffman(C:conjunto;f:vectFrec)**devuelve** árbol

{**Pre:** C es el conjunto de caracteres y f es el vector de frecuencias}

{**Post:** z es el árbol de un código libre de prefijos óptimo para (C,f)}

creaVacía(Q);

para todo x **en** C **hacer**

 inserta(Q,<x,f[x]>)

fpara;

para i:=1 **hasta** |C|-1 **hacer**

 <x,fx>:=primero(Q);

 borra(Q);

 <y,fy>:=primero(Q);

 borra(Q);

 fz:=fx+fy;

 z:=creaÁrbol(raíz=>fz,hijoIzq=>x,hijoDer=>y);

 inserta(Q,<z,fz>)

fpara;

<z,fz>:=primero(Q);

borra(Q);

devuelve z

Q:colaPri;

i,fx,fy,fz:entero;

z,x,y:árbol