


Universidad Técnica Federico Santa María - Departamento de Informática
 


Estructura de Datos


Códigos de Huffman

Prof.: Mauricio Solar Prof.: Lorna Figueroa

Primer Semestre,
2010

1





Universidad Técnica Federico Santa María - Departamento de Informática
 

Compresión de Archivos

- Los algoritmos estudiados hasta ahora han sido diseñados, en general, para que utilicen el menor tiempo posible, dejando la eficiencia de espacio en un segundo plano.
- La compresión de archivos lo considera en forma inversa: utilizar el menor espacio posible, sin considerar el tiempo empleado.

2




Universidad Técnica Federico Santa María - Departamento de Informática
 

Compresión de Archivos

Ejemplo: **Codificación por Longitud de series**


- El tipo mas simple de redundancia que se puede encontrar en un archivo son las series de caracteres repetidos:
AAAABBBBAABBBBCCCCCCCCDABCBAAABBBBCCCD
- Esta cadena se puede codificar de forma mas compacta reemplazando cada repetición de caracteres por un solo carácter repetido precedido del numero de veces que este se repite:
4A3BAA5B8CDABCB3A4B3CD

3



Universidad Técnica Federico Santa María - Departamento de Informática

Compresión de Archivos



Ejemplo: Codificación por Longitud de series


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

AAAABBBBAABBBBBBCCCCCCCCDABCBAAABBBBCCCD


- Esta cadena se puede codificar de forma mas compacta reemplazando cada repetición de caracteres por un solo carácter repetido producto del numero de veces que este se repite:

4A3BAASB8CDABCB3A4B3CD

[illegible]



Universidad Técnica Federico Santa María - Departamento de Informática




Compresión de Archivos

Codificación de Longitud Variable

- 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.
- Ejemplo: “ABRACADABRA”,
 - su representación en código binario está dada por 5 bits de i que reproducen la i-esima letra del alfabeto (0 para los blancos)


00001	00010	10010	00001	00011	00001	01000	00001	00010	10010	00001
A	B	R	A	C	A	D	A	B	R	A

Obs.: “D” aparece una vez en la cadena mientras que “A” aparece cinco, sin embargo ambos caracteres necesitan el mismo numero de bits.



Universidad Técnica Federico Santa María - Departamento de Informática

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:

$A \rightarrow 0, B \rightarrow 1, R \rightarrow 01, C \rightarrow 10, D \rightarrow 11$

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 15 bits en lugar de los 55 de la anterior.
- Depende de los espacios en blancos pues sin ellos se podría decodificar de forma incorrecta,
 - agregar 10 bits mas en delimitadores no hace que sea mas grande que la original

7



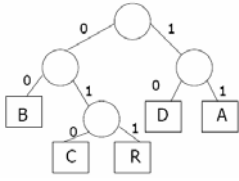
Universidad Técnica Federico Santa María - Departamento de Informática

Compresión de Archivos




- Una forma fácil de representar el código es mediante un árbol.
- Cualquier árbol de M hojas es capaz de representar un mensaje de M caracteres diferentes:

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




8



Universidad Técnica Federico Santa María - Departamento de Informática

Compresión de Archivos




- Luego, se tiene:

$A \rightarrow 11, B \rightarrow 00, R \rightarrow 011, C \rightarrow 010, D \rightarrow 10$

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

9




Universidad Técnica Federico Santa María - Departamento de Informática

Compresión de Archivos

- La representación por un árbol garantiza que el código es unívocamente decodificable a partir del árbol.
- Problema: se pueden obtener distintos árboles para un mismo mensaje dependiendo de cómo haya sido codificado.
- Existe una forma de construir un árbol que entregue 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**

10




Universidad Técnica Federico Santa María - Departamento de Informática

Aplicaciones de los Árboles Binarios – Códigos de Huffman

- Uno de los primeros códigos que se usan en algoritmos de compresión de la información o criptografía, etc.
- Cada letra del alfabeto es una hoja de un árbol binario y en el que los caminos más largos a las hojas se intentan sean los de las letras de menor frecuencia de aparición.
- Cada nodo, excepto la raíz, está etiquetado con un 0 o un 1;
 - 0 indica una rama izquierda,
 - 1 indica una rama derecha.

11




Universidad Técnica Federico Santa María - Departamento de Informática

Aplicaciones de los Árboles Binarios – Códigos de Huffman


- Una letra queda descrita por un camino de 0's y 1's con la propiedad de que todas y cada una de las letras tendrán secuencias de 0's y 1's siempre diferentes.
- Se tiene la propiedad de prefijos,
 - equivale, en la representación arbórea de las letras, a que cada letra sea una hoja.

12



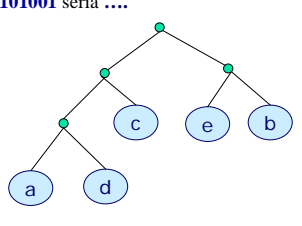
Universidad Técnica Federico Santa María

Departamento de Informática




Aplicaciones de los ArbolB – Códigos de Huffman

Por ejemplo, si $a \rightarrow 000$; $b \rightarrow 11$, $c \rightarrow 01$ y $d \rightarrow 001$, la secuencia **1101001** sería




13



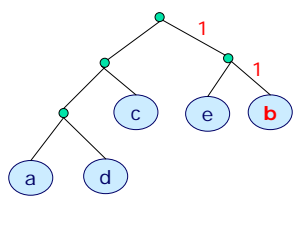
Universidad Técnica Federico Santa María

Departamento de Informática




Aplicaciones de los ArbolB – Códigos de Huffman

Por ejemplo, si $a \rightarrow 000$; **$b \rightarrow 11$** , $c \rightarrow 01$ y $d \rightarrow 001$, la secuencia **1101001** sería **b....**




14



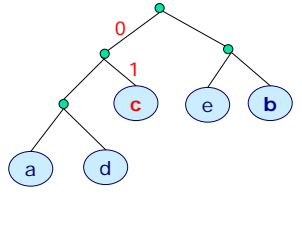
Universidad Técnica Federico Santa María

Departamento de Informática



Aplicaciones de los ArbolB – Códigos de Huffman

Por ejemplo, si $a \rightarrow 000$; **$b \rightarrow 11$** , **$c \rightarrow 01$** y $d \rightarrow 001$, la secuencia **1101001** sería **bc....**



15

Universidad Técnica Federico Santa María - Departamento de Informática

Aplicaciones de los Árboles - Códigos de Huffman

Por ejemplo, si $a \rightarrow 000$; $b \rightarrow 11$, $c \rightarrow 01$ y $d \rightarrow 001$, la secuencia **1101001** sería **bcd....**

16

Universidad Técnica Federico Santa María - Departamento de Informática

Aplicaciones de los Árboles - Códigos de Huffman

Por ejemplo, si $a \rightarrow 000$; $b \rightarrow 11$, $c \rightarrow 01$ y $d \rightarrow 001$, la secuencia **1101001** sería **bcd**.

- El problema de los códigos de Huffman es encontrar el reparto de códigos según la frecuencia de aparición de cada letra, de forma que el código resultante sea lo más corto posible.

17


Universidad Técnica Federico Santa María - Departamento de Informática

Códigos de Huffman

Construcción del código de Huffman:


- Contar el número de veces que se repite un carácter (frecuencia de aparición), en el mensaje que se va a codificar.
- Construir el árbol de abajo hacia arriba, de acuerdo con las frecuencias.
 - Al construir el árbol se considera primero como un árbol binario, luego como un árbol de codificación.

18



Universidad Técnica Federico Santa María - Departamento de Informática

Códigos de Huffman




Construcción del árbol:

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 más pequeñas y se unen en un nodo cuyos hijos serán los unidos; la frecuencia del nodo nuevo será la suma de 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 elección genera árboles distintos pero equivalentes desde el punto de vista de la optimalidad.

19




Universidad Técnica Federico Santa María - Departamento de Informática

Descripción del problema




- Se tiene un **archivo de entrada**.
- Asumir que el archivo está compuesto de **bytes** (enteros de 8 bits sin signo).
- El problema consiste en **codificar** el archivo de entrada utilizando el menor número posible de bits (comprimir).
- El algoritmo que resuelve este problema, es el conocido *algoritmo de Huffman*.

20



Universidad Técnica Federico Santa María - Departamento de Informática


Solución del problema



La solución se obtiene aplicando cuatro fases:


1. Crear un **vector de frecuencias** de aparición de cada byte en el archivo de entrada.
2. Crear el **árbol óptimo de codificación** de Huffman a partir del vector de frecuencias.
3. Crear una **tabla de codificación** a partir del árbol de codificación.
4. La **codificación** propiamente tal.

21

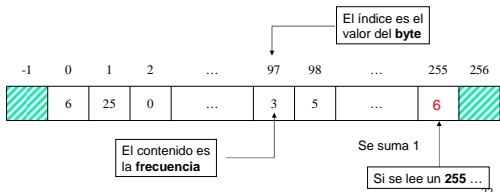



Universidad Técnica Federico Santa María - Departamento de Informática

Vector de frecuencias




- Se necesita conocer la frecuencia de aparición de cada byte en el archivo de entrada.
 - se tiene que hacer una primera lectura del archivo de entrada.
- El objetivo es crear un vector de la forma:



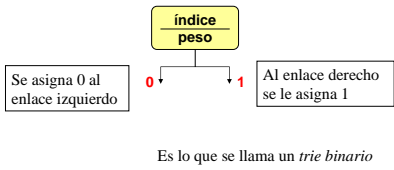



Universidad Técnica Federico Santa María - Departamento de Informática

Árbol óptimo de codificación (1)




- Se crea utilizando el **algoritmo de Huffman**.
- Este algoritmo se puede clasificar como *algoritmo voraz*.
- Las **entradas** del algoritmo son los bytes en el archivo de entrada.
- Trabaja con un **conjunto de árboles** (un **bosque**).
- Nodo** de un árbol de Huffman:






Universidad Técnica Federico Santa María - Departamento de Informática


Árbol óptimo de codificación (2)



- Los **nodos hoja** del árbol de Huffman serán los bytes leídos en el archivo de entrada. En ese caso:
 - índice** = valor del byte (entre 0 y 255)
 - peso** = frecuencia del byte
- A partir del vector de frecuencias se construye un **bosque** de árboles que sólo tienen un nodo.
 - Cada nodo representará a un byte, con su **índice (valor)** y su **peso (frecuencia)**.
- Un byte sólo está presente en el bosque inicial si aparece en el archivo de entrada con una frecuencia distinta de cero.

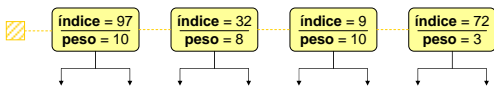


Universidad Técnica Federico Santa María · Departamento de Informática




Árbol óptimo de codificación (3)

- El “bosque” inicial podría consistir, por ejemplo, en una lista de nodos.




- Ahora se debe extraer los dos nodos con **menor peso**.
→ ¿Una lista es la mejor opción posible?

25

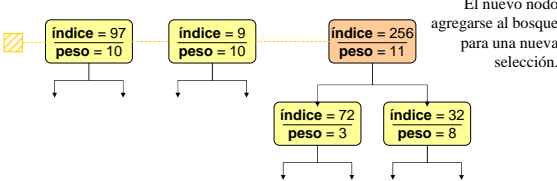


Universidad Técnica Federico Santa María · Departamento de Informática




Árbol óptimo de codificación (4)


- A partir de los dos nodos con menor peso extraídos se construye un nuevo nodo.
 - Su peso será la suma de los pesos de sus hijos.
- El índice no importa aún; sólo es importante para la construcción de la tabla de codificación. A partir de ahora tomará los valores 256, 257, 258, etc.



26

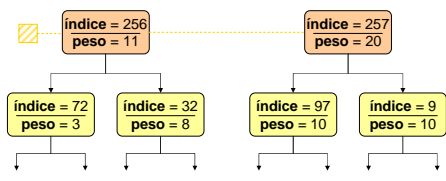


Universidad Técnica Federico Santa María · Departamento de Informática




Árbol óptimo de codificación (5)

- El proceso es iterativo:




27



Universidad Técnica Federico Santa María · Departamento de Informática

Árbol óptimo de codificación (6)



- Hasta que quede un único nodo:

índice = 258

peso = 31

índice = 256

peso = 11

índice = 257

peso = 20

índice = 72

peso = 3

índice = 32

peso = 8


índice = 97

peso = 10

índice = 9


peso = 10

28



Universidad Técnica Federico Santa María · Departamento de Informática

Árbol óptimo de codificación (7)



- En la siguiente extracción, el bosque queda vacío y se obtiene el árbol de Huffman:

índice = 257

peso = 31

índice = 255

peso = 11

índice = 256

peso = 20

índice = 72

peso = 3

índice = 32

peso = 8

índice = 97

peso = 10


índice = 9

peso = 10

¿Codificación de 97?


10

29




Universidad Técnica Federico Santa María · Departamento de Informática

Árbol óptimo de codificación (8)




- Son posibles varios árboles de codificación.
- Todos son óptimos en el sentido de que entregan una codificación con el menor número de bits.
- La mejor estructura para almacenar los nodos del bosque (conjunto de candidatos) es una **cola de prioridad**, donde la **prioridad** es el **peso** del nodo.
- Esto permite obtener el mínimo código de una manera eficaz ($O(\log n)$).

30




Universidad Técnica Federico Santa María - Departamento de Informática

Árbol óptimo de codificación (9)




- El análisis de Huffman dice que si el número de entradas es C , el número máximo de nodos del árbol de codificación es $2C - 1$.
- Esto permite utilizar como cola de prioridad un **montículo binario** (se implementa con un array).
- En este caso C es, como máximo, 256.
- El montículo binario puede tener un tamaño seguro de $(2 * 256 - 1)$.

31




Universidad Técnica Federico Santa María - Departamento de Informática

Tabla de codificación (1)




- La tabla de codificación es un paso intermedio entre el árbol de codificación y la verdadera codificación.
- Se construye para que la codificación sea más sencilla y rápida.
- Sin embargo, para construirla es necesario que cada nodo del árbol de Huffman pueda **referenciar a su padre**, y que almacene información sobre el **tipo de hijo** (0 ó 1).

32



Universidad Técnica Federico Santa María - Departamento de Informática

Tabla de codificación (2)




- Un nodo de la tabla de codificación puede contener la información siguiente:

peso	índice padre	tipoHijo


- La tabla consiste en un array que sirve para indexar nodos de codificación.
- El tamaño máximo de este array lo indica el índice del nodo raíz del árbol Huffman.

33

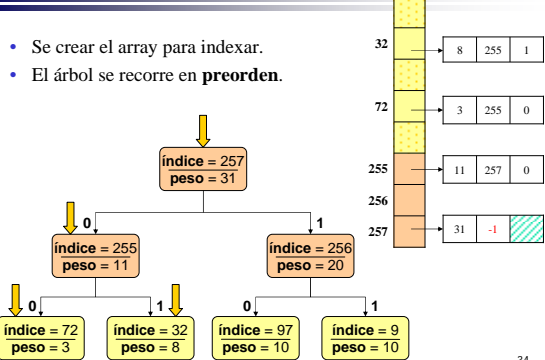


Universidad Técnica Federico Santa María - Departamento de Informática


Tabla de codificación (3)



- Se crear el array para indexar.
- El árbol se recorre en **preorden**.




34



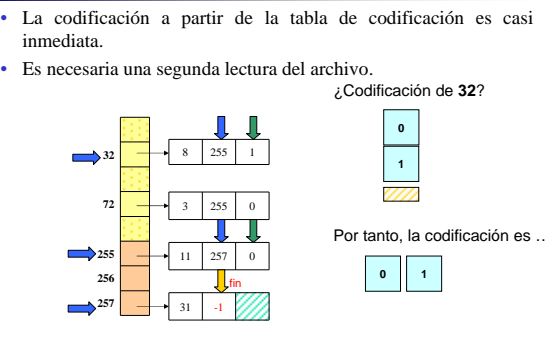
Universidad Técnica Federico Santa María - Departamento de Informática

Codificación




- La codificación a partir de la tabla de codificación es casi inmediata.
- Es necesaria una segunda lectura del archivo.

¿Codificación de 32?




Por tanto, la codificación es ...

35




Universidad Técnica Federico Santa María - Departamento de Informática

Complejidad




- La complejidad del algoritmo que crea el árbol de Huffman es la de una extracción en una cola de prioridad, que tiene complejidad $O(\log n)$.
- Por tanto, la complejidad de todo el algoritmo, que es iterativo, será $O(n \log n)$.

36



Universidad Técnica Federico Santa María - Departamento de Informática

Aplicación: Códigos de Huffman




EJEMPLO:

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


	a	b	c	d	e	f
Frecuencia (en miles)	45	13	12	16	9	5

37



Universidad Técnica Federico Santa María - Departamento de Informática


Aplicación: Códigos de Huffman



- Codificando este archivo mediante el uso del código de longitud variable, y asignando la menor cadena de bits al carácter que más se repite, se obtiene:


	a	b	c	d	e	f
Código longitud var.	0	101	100	111	1101	1100

38



Universidad Técnica Federico Santa María - Departamento de Informática

Aplicación: Códigos de Huffman



- Construyendo el árbol...

f: 5

e: 9

c: 12

b: 13

d: 16

a: 45

c: 12

b: 13

14

d: 16

a: 45

0

1

f: 5

e: 9

39

Universidad Técnica Federico Santa María - Departamento de Informática

Aplicación: Códigos de Huffman

- Construyendo el árbol...

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

```

graph TD
    14((14)) ---|0| f[f: 5]
    14 ---|1| e[e: 9]
    25((25)) ---|0| c[c: 12]
    25 ---|1| b[b: 13]
    45((45)) ---|0| 14
    45 ---|1| 25
  
```

40

Universidad Técnica Federico Santa María - Departamento de Informática

Aplicación: Códigos de Huffman

- Construyendo el árbol...

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

```

graph TD
    55((55)) ---|0| 45((45))
    55 ---|1| 14((14))
    14 ---|0| c[c: 12]
    14 ---|1| b[b: 13]
    c ---|0| f[f: 5]
    c ---|1| e[e: 9]
    b ---|1| d[d: 16]
  
```

41

Universidad Técnica Federico Santa María - Departamento de Informática

Aplicación: Códigos de Huffman


- Construyendo el árbol...

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

```


graph TD
    55((55)) ---|0| 25((25))
    55 ---|1| 30((30))
    25 ---|0| c[c: 12]
    25 ---|1| b[b: 13]
    c ---|0| f[f: 5]
    c ---|1| e[e: 9]
    b ---|1| d[d: 16]
  
```

42

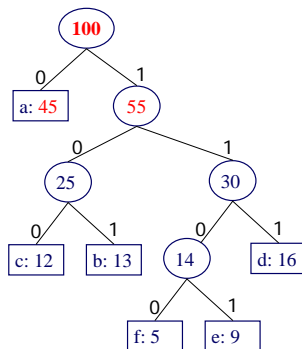


Universidad Técnica Federico Santa María - Departamento de Informática


Aplicación: Códigos de Huffman



- Finalmente:




43




Universidad Técnica Federico Santa María - Departamento de Informática

Aplicación: Códigos de Huffman




- Los caracteres 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:
 - La longitud del mensaje codificado es igual a la longitud ponderada del camino externo del árbol de frecuencias de Huffman.
 - 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.

44



Universidad Técnica Federico Santa María - Departamento de Informática

Consideraciones finales



- Para **decodificar** es necesario almacenar en el archivo codificado más información que los bits de los códigos.
- Evidentemente interesa que esa información ocupe el menor espacio posible.
- ¿Quizás la tabla de codificación?

45



Bibliografía



- **Estructuras de datos en Java**
Mark Allen Weiss
Editorial: Addison-Wesley
- **Técnicas de Diseño de Algoritmos**
Rosa Guerequeta y Antonio Vallecillo
- **Compresión de Archivos**
Gabriel Omaña
