

textos sobre programación

Artículos con Clase

Artículos con Clase

Estilo:

0000646009

Visitas desde 2002-04-27

0090

Usuarios en línea

- [Con Clase](#)
- [C++](#)
- [Win API 32](#)
- [HTML y CSS](#)
- [Gráficos](#)
- [MySQL](#)
- [Artículos](#)
- [Listas](#)

- [Comentario](#)

- [Novedades](#)
- [Colabora](#)

- 
- [Algoritmos](#)
    - [Compresión de Huffman](#)
      - [Generalidades](#)
      - [Mecanismo del algoritmo](#)
      - [Veamos un ejemplo](#)
      - [Programas en C](#)
    - [Árboles B](#)

- [Matemáticos](#)
- [Ordenación](#)
- [Gráficos](#)
- [Windows \(José Navarro\)](#)
- [Juegos](#)
- [Multimedia](#)
- [Opinión](#)
- [Encriptación](#)
- [Comunicación](#)

- 
- [Firmar libro](#)
  - [Ver libro](#)

# Algoritmo de compresión de Huffman

^  
—

## Generalidades

^  
—

Se trata de un algoritmo que puede ser usado para compresión o encriptación de datos.

Este algoritmo se basa en asignar códigos de distinta longitud de bits a cada uno de los caracteres de un fichero. Si se asignan códigos más cortos a los caracteres que aparecen más a menudo se consigue una compresión del fichero.

Esta compresión es mayor cuando la variedad de caracteres diferentes que aparecen es menor. Por ejemplo: si el texto se compone únicamente de números o mayúsculas, se conseguirá una compresión mayor.

Para recuperar el fichero original es necesario conocer el código asignado a cada carácter, así como su longitud en bits, si ésta información se omite, y el receptor del fichero la conoce, podrá recuperar la información original. De este modo es posible utilizar el algoritmo para encriptar ficheros.

## Mecanismo del algoritmo



- Contar cuantas veces aparece cada carácter en el fichero a comprimir. Y crear una lista enlazada con la información de caracteres y frecuencias.
- Ordenar la lista de menor a mayor en función de la frecuencia.
- Convertir cada elemento de la lista en un árbol.
- Fusionar todos estos árboles en uno único, para hacerlo se sigue el siguiente proceso, mientras la lista de árboles contenga más de un elemento:
  - Con los dos primeros árboles formar un nuevo árbol, cada uno de los árboles originales en una rama.
  - Sumar las frecuencias de cada rama en el nuevo elemento árbol.
  - Insertar el nuevo árbol en el lugar adecuado de la lista según la suma de frecuencias obtenida.
- Para asignar el nuevo código binario de cada carácter sólo hay que seguir el camino adecuado a través del árbol. Si se toma una rama cero, se añade un cero al código, si se toma una rama uno, se añade un uno.
- Se recodifica el fichero según los nuevos códigos.

## Veamos un ejemplo



Tomemos un texto corto, por ejemplo:

"ata la jaca a la estaca"

1) Contamos las veces que aparece cada carácter y hacemos una lista enlazada:

'(5), a(9), c(2), e(1), j(1), l(2), s(1), t(2)

2) Ordenamos por frecuencia de menor a mayor

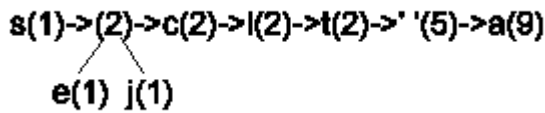
e(1), j(1), s(1), c(2), l(2), t(2), '(5), a(9)

3) Consideremos ahora que cada elemento es el nodo raíz de un árbol.

**e(1)->j(1)->s(1)->c(2)->l(2)->t(2)->'(5)->a(9)**

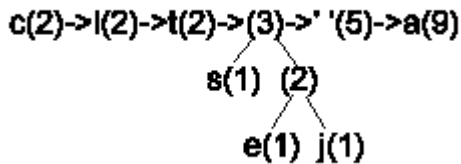
Estructura inicial

4) Fundimos los dos primeros nodos (árboles) en un nuevo árbol, sumamos sus frecuencias y lo colocamos en el lugar correspondiente:



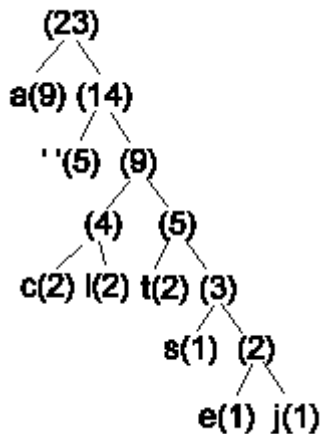
Estructura después de primera fusión

Y sucesivamente:



Estructura después de varias fusiones

El resultado final es:



Resultado final de la estructura

5) Asignamos los códigos, las ramas a la izquierda son ceros, y a la derecha unos (por ejemplo), es una regla arbitraria.

a	'	c	l	t	s	e	j
0 10	1100	1101	1110	11110	111110	111111	

6) Y traducimos el texto:

a	t	a	'	l	a	'	j	a	c	a	'	a	'	l	a	'	e	s	t	a	c	a
0 1110	0 10	1101	0 10	111111	0 1100	0 10	0 10	1101	0 10	111110	11110	1110	0 1100	0								

Y sólo queda empaquetar los bits en grupos de ocho, es decir en bytes:

01110010	11010101	11111011	00010010	11010101	11110111	10111001	10000000
0x72	0xD5	0xFB	0x12	0xD5	0xF7	0xb9	0x80

En total ocho bytes, y el texto original tenía 23.


Pero no nos engañemos, también hay que almacenar la información relativa a la codificación, por lo que se puede ver que para textos cortos no obtendremos mucha reducción de tamaño.

# Programas en C



Bueno, y ahora la implementación en C.

Hay dos programas:

Nombre	Fichero	Fecha	Tamaño	Contador	Descarga
 Compresor	Compres.zip	2001-02-01	10060 bytes	6545	<a href="#">Descargar</a>

Nombre	Fichero	Fecha	Tamaño	Contador	Descarga
 Descompresor	Decomp.zip	2001-02-01	4850 bytes	2703	<a href="#">Descargar</a>

## Comentarios de los usuarios (2)



**miguel**

2013-11-24 22:30:48

como puedo probar el programa de huffman para ver lo que hace



**Mariano Ruiz**

2018-07-20 02:10:08

Excelente artículo que me permitió hace varios años ya entender el algoritmo, siempre vigente. Acabo de subir a Github un fork de ambas implementaciones (compresor y descompresor):

<https://github.com/mrsarm/compres>

Tiene estas modificaciones:

- Código portado a C: mínimos cambios ya que el código original no es C sino C++, pero no hacía uso de programación orientada a objetos, solo uso de referencias que no son compatibles con C.
- Fix error en cómo se leía el archivo de entrada que causaba que no se pudiera usar para comprimir archivos binarios o con más de 128 caracteres diferentes.
- Archivos re-encodeados a UTF-8 en vez del viejo encoding ASCII ISO-8859-1, que hacía in-leíble los comentarios con caracteres diacríticos en cualquier editor moderno no Windows.

---

  Buscar en: 

suministrado por [FreeFind](#)



© Febrero de 2001, Salvador Pozo Coronado, [salvador@conclase.net](mailto:salvador@conclase.net)