ALGORITMOS Y ESTRUCTURA DE DATOS 2018

ALGORITMO DE HUFFMAN

1.1. Compresión de archivos mediante el algoritmo de Huffman

1.1.1. Introducción

Clase 10

Ir al video

Generalmente utilizamos múltiplos del byte para dimensionar diferentes cantidades de información. Por ejemplo: "tengo un disco rígido con más de 100 gigabytes de música" o "la base de datos de la compañia supera el terabyte de información"; o "no puedo enviarte el video por mail porque pesa 300 megabytes".

En cambio, si queremos dimensionar el caudal de información que se transmite por un canal durante una determinada cantidad de tiempo hablamos de "bits por unidad de tiempo"

Por ejemplo: "el servicio de video *on demand* requiere un ancho de banda no menor a 3 megabits por segundo" o "conseguí la canción que buscaba en mp3 *rippeada* a 320 kbps (kilobits por segundo)".

Los programas de compresión de información como WinRAR o WinZIP utilizan algoritmos que permiten reducir esas cantidades de bytes. Cuando *zippeamos* un archivo obtenemos un nuevo archivo que contiene la misma información que el anterior pero ocupa menos espacio en disco.

Aquí estudiaremos el algoritmo de Huffman, desarrollado por David A. Huffman y publicado en *A method for the construction of minimun redundancy codes*, en 1952. El algoritmo permite comprimir información basándose en una idea simple: utilizar menos de 8 bits para representar a cada uno de los bytes de la fuente de información original.

Por ejemplo, si utilizamos los bits 01 (dos bits) para representar el carácter 'A' (o su byte equivalente: 01000001), cada vez que aparezca una 'A' estaremos economizando 6 bits. Entonces cada 4 caracteres 'A' que aparezcan en el archivo ahorraremos 24 bits, es decir, 3 bytes. Luego, cuanto mayor sea la cantidad de caracteres 'A' que contenga el archivo o la fuente de información original, mayor será el grado de compresión que podremos consegir. A la secuencia de bits que utilizamos para recodificar a cada carácter la llamamos: código Huffman.

El algoritmo consiste en una serie de pasos a través de los cuales podremos construir los códigos Huffman que reemplazarán a los bytes de la fuente de información que vamos a comprimir; generando códigos más cortos para los caracteres más frecuentes y códigos más largos para los bytes que menos veces aparecen.

Alcance del ejercicio

Implementar un compresor de archivos basado en el algoritmo de Huffman es un excelente ejercicio que nos inducirá a aplicar los principales conceptos que estudiamos desde el comienzo: *arrays*, archivos y listas enlazadas.

El algoritmo utiliza un árbol binario para generar los códigos Huffman que reemplazarán a los bytes de la fuente de información original. El hecho de que aún no hayamos estudiado árboles binarios representa una excelente oportunidad para aplicar los conceptos de abstracción y encapsulamiento.

Es decir, este ejercicio persigue tres objetivos:

1. Estudiar y analizar el algoritmo de Huffman.
2. Desarrollar una aplicación que, basándose en este algoritmo, permita comprimir y descomprimir archivos.
3. Proveer implementaciones concretas para las interfaces de los objetos que vayamos a utilizaren el programa. Estas implementaciones estarán a cargo del alumno y constituyen el ejercicio en sí mismo.

1.1.2. El algoritmo de Huffman

1.1.2.1. Introducción

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 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. Por ejemplo, en el siguiente texto:

COMO COME COCORITO COME COMO COSMONAUTA

el carácter 'O' aparece 11 veces y el carácter 'C' aparece 7 veces. Estos son los caracteres que más veces aparecen y, por lo tanto, tienen la mayor probabilidad de ocurrencia.

En cambio, los caracteres 'I', 'N', 'R', 'S' y 'U' aparecen una única vez; esto significa que la probabilidad d hallar en el archivo alguno de estos caracteres es muy baja.

Para codificar cualquier carácter se necesitan 8 bits (1 byte). Sin embargo, supongamos que logramos encontrar una combinación única de 2 bits con la cual codificar al carácter 'O', una combinación única de 3 bits con la cual codificar al carácter 'M' y otra combinación única de 3 bits para codificar al carácter 'C'.

Byte o Carácter	Codificación
O	01
M	001
C	000

Si esto fuese así, entonces para codificar los primeros 3 caracteres del texto anterior alcanzará 1 byte, lo que nos dará una tasa de compresión del 66.6%.

Carácter	C	O	M	...
Byte	000	01	001	...

Ahora, el byte 00001001 representa la secuencia de caracteres ‘C’, ‘O’, ‘M’; pero esta información sólo podrá ser interpretada si conocemos los códigos binarios que utilizamos para recodificar los bytes originales. De lo contrario, la información no se podrá recuperar.

Para obtener estas combinaciones de bits únicas, el algoritmo de Huffman propone seguir una serie de pasos a través de los cuales obtendremos un árbol binario llamado: arbol Huffman. Luego, las hojas del árbol representarán a los diferentes caracteres que aparecen en el archivo y los caminos que se deben recorrer para llegar hasta esas hojas representarán la nueva codificación del carácter.

A continuación, analizaremos, paso a paso, el algoritmo que nos permitirá obtener el árbol y los códigos Huffman que corresponden a cada uno de los caracteres del texto propuesto más arriba.

1.1.2.2. El algoritmo paso a paso

Paso 1: Determinar la cantidad de veces que aparece cada carácter.

El primer paso será contar cuantas veces aparece cada carácter (o byte) dentro del archivo; y para esto debemos tener en cuenta las siguientes consideraciones.

Un byte consiste en un conjunto de 8 bits que pueden combinarse de 2^8=256 maneras diferentes. Esto significa que sea cual fuere la cantidad de bytes que contenga un archivo, éstos podrán clasificarse entre, no más de, 256 variantes. Por supuesto, algunas más frecuentes que otras.

Entonces, para contar cuantas veces aparece cada byte (cada variante) dentro del archivo utilizaremos una tabla (*array*) de 256 filas. El índice *i* del *array* representará la *i-ésima* variante.

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

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

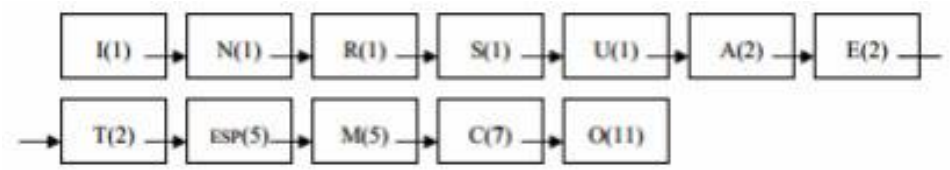
Conociendo cuantas veces aparece cada carácter, tenemos que crear una lista enlazada y ordenada as-cendentemente por dicha cantidad. Primero los caracteres menos frecuentes y luego los que tienen mayor

probabilidad de aparecer y, si dos caracteres ocurren igual cantidad de veces, entonces colocaremos primero

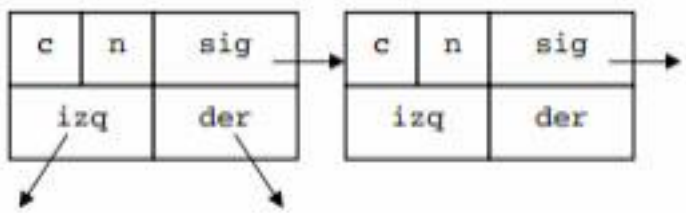
al que tenga menor valor numérico. Por ejemplo: los caracteres 'I', 'N', 'R', 'S' y 'U' aparecen una sola vez y tie-nen la misma probabilidad de ocurrencia entre sí; por lo tanto, en la lista ordenada que veremos a conti-nuación los colocaremos ascendentemente según su valor numérico o código ASCII.

Paso 2: Crear una lista enlazada.

Conociendo cuantas veces aparece cada carácter, tenemos que crear una lista enlazada, ordenada ascendentemente por dicha cantidad. Primero los caracteres menos frecuentes; 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; tienen la misma probabilidad de ocurrencia entre sí. Por lo tanto, en la lista ordenada que veremos a continuación los colocaremos de menor a mayor según su valor numérico o código ASCII.



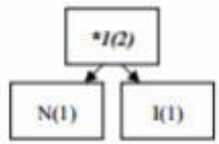
Los nodos de la lista tendrán la siguiente estructura:



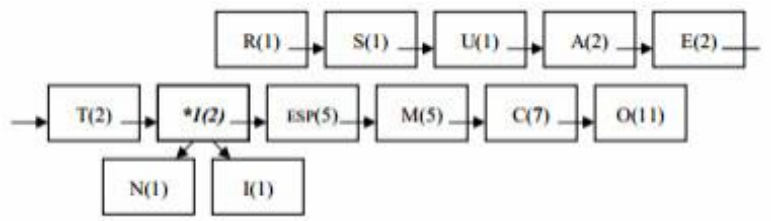
El campo `c` representa al carácter (o byte) propiamente dicho; y el campo `n` indica la cantidad de veces que que aparece el carácter `c` dentro del archivo. El campo `sig` es la referencia al siguiente nodo de la lista enlazada. Los campos `izq` y `der`, más adelante, nos permitirán implementar el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

Paso 3 - Convertir la lista enlazada en un árbol Huffman.

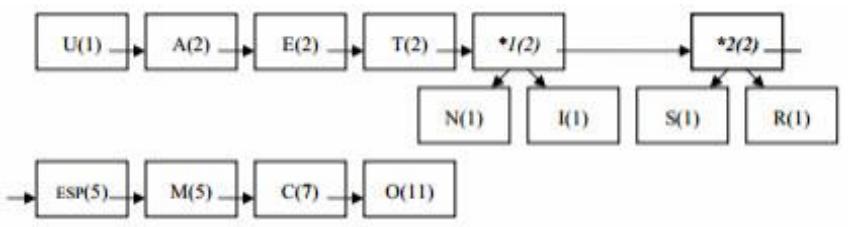
Vamos a generar el árbol Huffman tomando "de a pares" los nodos de la lista. Esto lo haremos de la siguiente manera: sacamos los dos primeros nodos y los utilizamos para crear un pequeño árbol binario, cuya raíz será un nuevo nodo que identificaremos con el carácter ficticio $*_1$ (léase "asterisco uno") y una cantidad de ocurrencias `n` 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.



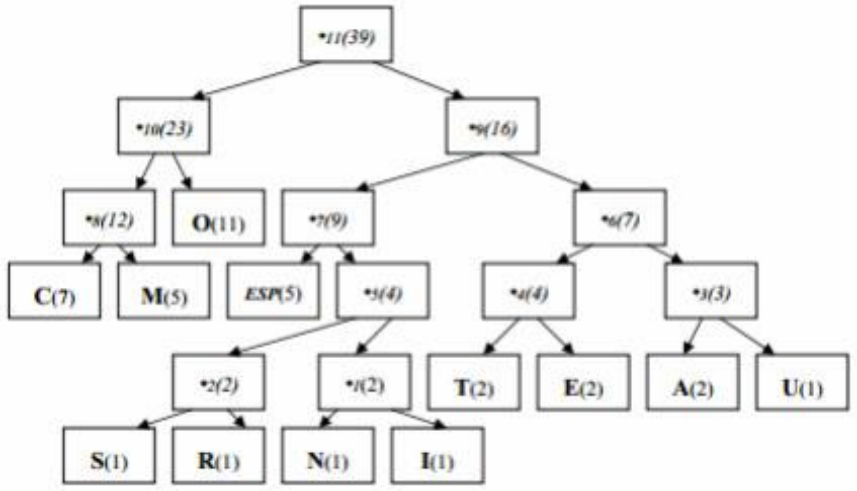
Luego insertamos en la lista al nuevo nodo $*_1$ respetando el criterio de ordenamiento que mencionamos más arriba. Si en la lista existe un nodo con la misma cantidad de ocurrencias, que en este caso es 2, la inserción la haremos a continuación de éste.



Repetimos la operación procesando los primeros dos 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 tendrá una cantidad de ocurrencias igual al tamaño del archivo que queremos comprimir.



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

Paso 4 - Asignación de los códigos Huffman

El siguiente paso será asignar un código Huffman a cada uno de los caracteres reales que, como vimos, se encuentran ubicados en las hojas del árbol.

Para esto debemos considerar el camino que une a la raíz del árbol con cada una de las hojas. 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.

Podemos verificar que la longitud del código que el árbol Huffman le asigna a cada carácter es inversamente proporcional a su cantidad de ocurrencias. Los caracteres más frecuentes reciben códigos más cortos y los que son menos frecuentes reciben códigos más largos.

	Carácter	Contador	Código
0			
:			
32	ESP	5	100
:			
65	A	2	1110
:			
67	C	7	000
:			
69	E	2	1101
:			
73	I	1	10111

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

Longitud máxima de un código Huffman

El lector podrá pensar que, en el peor de los casos, el código Huffman más largo que se le asignará a un carácter será de 8 bits; porque esta es la cantidad original de bits con que se representa a cada carácter. Sin embargo esto no es así.

La longitud máxima que puede alcanzar un código Huffman dependerá de la cantidad de símbolos que componen el alfabeto con el que esté codificada la información. De modo que, si *n* es la cantidad de símbolos que componen un alfabeto entonces, en el peor de los casos, el código Huffman que se asignará a un carácter podrá a tener hasta: *n*/2 bits.

En nuestro caso el alfabeto está compuesto por cada una de las 256 variaciones que admite un byte; por lo tanto, tenemos que estar preparados para trabajarcon códigos de, a lo sumo, 256/2 = 128 bits.

El hecho de asignar códigos de menos de 8 bits a los caracteres más frecuentes compensa la posibilidad de que, eventualmente, se utilicen más de 8 bits para codificar los caracteres que menos veces aparecen.

Paso 5 - Codificación y decodificación del contenido.

Codificación (compresión)

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

Carácter	C	O	M	O	[ESP]	C	…
Código	000	01	001	01	100	000	…
Byte	00001001			01100000			…

Decodificación (descompresión)

Para descomprimir un archivo necesitamos disponer del árbol Huffman que se utilizó para su codificación; sin el árbol la información no se podrá recuperar. Por este motivo el algoritmo de Huffman también se considera un método de encriptación.

Supongamos que, de algún modo, podemos reconstituir el árbol Huffman; entonces el algoritmo para descomprimir y restaurar el archivo original es el siguiente:

1. Rearmar el árbol Huffman y posicionarnos en la raíz.
2. Recorrer, “bit a bit” el archivo comprimido. Si leemos un 0 descendemos un nivel del árbol y nos posicionamos en su hijo izquierdo. En cambio, si leemos un 1 descendemos un nivel y nos posicionamos en su hijo derecho.

3. Repetimos el paso 2 hasta llegar a una hoja. Esto nos dará la pauta de que hemos decodificado un carácter y lo podremos grabar en el archivo que estamos restaurando.

1.1.3. Programa compresor/descompresor de archivos

El programa debe recibir, en línea de comandos, el nombre el archivo con el que va a trabajar. Si el nombre termina con el sufijo `.huf` significa que se trata de un archivo comprimido, cuyo contenido debe procesar para recomponer el archivo original. En cualquier otro caso se considera que se trata de un archivo original, cuyo contenido se deberá comprimir y almacenar en un nuevo archivo, cuyo nombre será el mismo nombre del archivo seguido de la extensión `.huf`.

Proceso de compresión

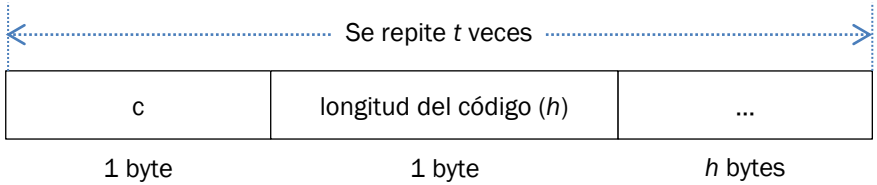
Proceso de descompresión

Estructura del archivo comprimido

El archivo comprimido, con extensión `.huf`, se compone de cuatro partes:

1. Al inicio contiene 1 byte indicando cuántas hojas conforman el árbol (*t*).
2. Luego *t* registros cuya estructura se explica más abajo.
3. A continuación 8 bytes (`long`) que indican cuantos caracteres fueron codificados; es decir: la longitud, en bytes, del archivo original.
4. Finalmente contiene la información codificada y comprimida.

Para almacenar el árbol Huffman se almanenarán, al inicio del archivo, *t* registros con la siguiente estructura:



Si consideramos que el árbol Huffman que se generó tiene exactamente *t* hojas entonces el archivo `.huf` comenzará con *t* registros, cada uno de los cuales representa a cada hoja (carácter o byte) y respeta el siguiente patrón:

- 1 byte con el carácter (o valor numérico) que representa la hoja del árbol.
- 1 byte indicando la longitud del código Huffman que se le asignó al carácter.
- *h* bytes que contienen los bits (caracteres '1' o '0') que componen el código Huffman del carácter.

A continuación de estos *t* registros, se ubicarán los 8 bytes que describen la longitud del archivo original y, luego, los bits de los códigos Huffman de cada uno de los caracteres que fueron codificados.