Artículo de investigación sobre el algoritmo de compresión de Huffman

1st Fabio López
Facultad de Ciencias e Ingeniería
Pontificia Universidad Católica del Perú
Lima, Perú
a20155912@pucp.edu.pe

2nd Raúl Jeri
Facultad de Ciencias e Ingeniería
Pontificia Universidad Católica del Perú
Lima, Perú
raul.jeri@pucp.edu.pe

3rd Juan Diego Veliz
Facultad de Ciencias e Ingeniería
Pontificia Universidad Católica del Perú
Lima, Peru
a20170089@pucp.edu.pe

***Abstract*—Data compression is the ability to encoding information using fewer bits than is needed. This technology reduces the amount of data used to represent any content with or without quality loss. There are several techniques available for use. In this paper we have analyzed the algorithm of Huffman and its application in the understanding of simple text files as a practice of applying the Binary Tree abstract data type.**

***Index Terms*—Huffman, Compression, Encryption, Binary Tree, Abstract, Data, Type**

Introducción

Las sociedades actuales se han ido transformando debido al desarrollo y expansión de la informática. Esta ingeniería ofreció, desde su origen, métodos para manipular virtual y eficientemente cualquier tipo de información. Sin embargo, la constante evolución de esta ha creado una rápida expansión en el volumen de la información a almacenar, tanto en discos duros como en servidores de Internet, y, consecuentemente, una necesidad relevante por comprimir datos. La compresión de datos, ***'Data compression'***, consiste en la habilidad de reducir la cantidad de almacenamiento. Esta compresión mencionada puede ser con pérdida o sin pérdida. Por un lado, la compresión con pérdida se identifica en los casos de compresión de fotografías, dado a que, en estas situaciones, los datos no necesitan ser almacenados perfectamente. Debido a esto, las fotos se comprimen con una técnica de compresión con pérdidas, distorsionando la calidad inicial de la imagen. Por otro lado, la compresión sin pérdida se convierte fundamental en situaciones en las que no puede permitirse que la información sea degradada, como por ejemplo en archivos de texto. Asimismo, cuando el propósito es archivar imágenes, videos y audios en su estado original, a menudo se usa compresión sin pérdidas. El algoritmo, o codificación, de Huffman propone el método más eficiente de compresión del tipo de representación prefija.

Este informe presenta, en primer lugar, una introducción a conceptos tales como el manejo de datos por parte del computador actual y la definición de la estructura de árbol binario, y, en segundo lugar, un análisis del algoritmo de Huffman que incluye las instrucciones de su especificación, implementación y posterior soporte matemático.

El problema que se pretende resolver con este algoritmo consiste en la compresión de la información acumulada en un centro educativo. En otros términos, se desean comprimir exámenes, documentos de matrícula, información de profesores y empleados, entre otros, en formato de texto. Esto es, precisamente, para luego ser almacenado en una base de datos que contenga información relevante para un año escolar.

Conceptos Importantes

Manejo de Datos

Actualmente, la gran mayoría de sistemas de computadoras de considerable demanda son las que se constituyen de codificación binaria. Esta codificación gobierna las variedades de funciones que realiza la computadora. Por tal motivo, toda información almacenada en este sistema debe ser representada por 0's y 1's.

Un archivo de texto nos muestra representaciones gráficas de caracteres, los cuales son comprendidos por la computadora como un código binario. (Véase Fig 1)

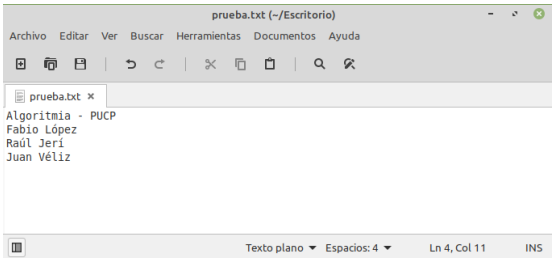

Fig. 1. Archivo de texto

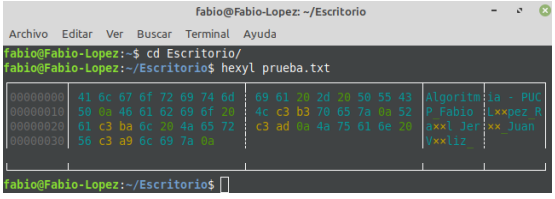

Fig. 2. Comprensión hexadecimal de archivo de texto por computadora

Para posibilitar el uso de caracteres en un computador, se implementó una codificación ASCII de 256 combinaciones que representa cada carácter. Estas combinaciones son posibles con un espacio de 8 bits. Por ende, cada letra, según el sistema de codificación de caracteres alfanuméricos ($ASCII$), comprende un espacio de memoria de 8 bits. (Véase Fig. 3)

Carácter	*Hexadecimal*	*Binario*
A	41	0100 0001
B	42	0100 0010
C	43	0100 0011
D	44	0100 0100
E	45	0100 0101
F	46	0100 0110
G	47	0100 0111
H	48	0100 1000
I	49	0100 1001
J	4A	0100 1010
K	4B	0100 1011
L	4C	0100 1100
M	4D	0100 1101
N	4E	0100 1110
O	4F	0100 1111
P	50	0101 0000
Q	51	0101 0001

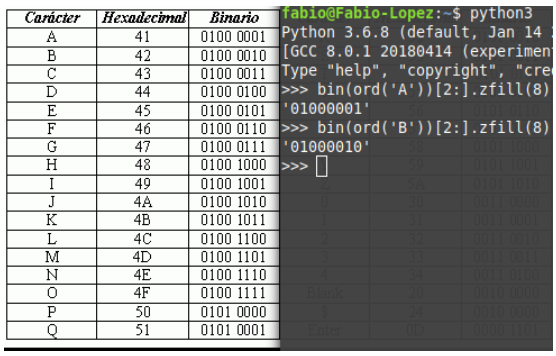

Fig. 3. Codificación ASCII

Debido a esta codificación se establece que el espacio de memoria de un archivo de texto depende de la cantidad de caracteres (letras, símbolos, saltos de páginas, espacios, tildes, etc) presentes en él. La fórmula que expresa la relación mencionada es la siguiente:

$$T \approx 8 * l \tag{1}$$

- T: tamaño del archivo en bits.
- l: cantidad de caracteres en el archivo.

Se considera que una letra con tilde ('á'), o una representación gráfica de caracteres combinados, es comprendida por algunos editores de texto como dos caracteres disjuntos (la tilde seguida de la letra), pese a existir una codificación ASCII específica para este caso. Esto explica tanto el valor aproximado mostrado en la $ecuación\ 1$, como también el resultado hexadecimal de las letras tildadas en la $Figura\ 2$.

Finalmente, dependiendo del editor de texto instalado y el sistema de codificación de caracteres del computador, un archivo de texto como el de la $Fig.\ 1$ posee un espacio de memoria base de 432 bits, es decir, de 54 bytes. Esto se explica dado a que se encuentran 54 caracteres presentes entre letras, espacios, saltos de línea, tildes y un carácter final de fichero.

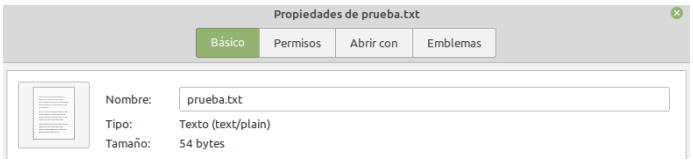

Fig. 4. Peso del archivo de texto

Estructura de Árbol Binario

El tipo de dato abstracto denominado Árbol Binario posee una configuración y arquitectura que posibilita una óptima inserción y recepción de datos. Esta clase cuenta con los atributos principales de valor, hijo izquierdo e hijo derecho. La función del atributo "valor" es almacenar un dato en el objeto. Por otra parte, las instancias "hijo izquierdo" e "hijo derecho" son punteros hacia otros objetos de la misma clase o punteros nulos dependiendo del caso. Esta estructura definida permite realizar operaciones con un tiempo de ejecución logarítmico en el peor de los casos:

$$T(n) = O(\log n) \tag{2}$$

De esta manera, se pueden construir variedades de árboles relacionados unos con otros gracias a las referencias que ofrecen las instancias de "hijo izquierdo" e "hijo derecho". En el presente informe, se denominará como $Árbol\ de\ Huffman$ a aquel objeto $Árbol\ Binario$ que contenga en su estructura información relevante al algoritmo.

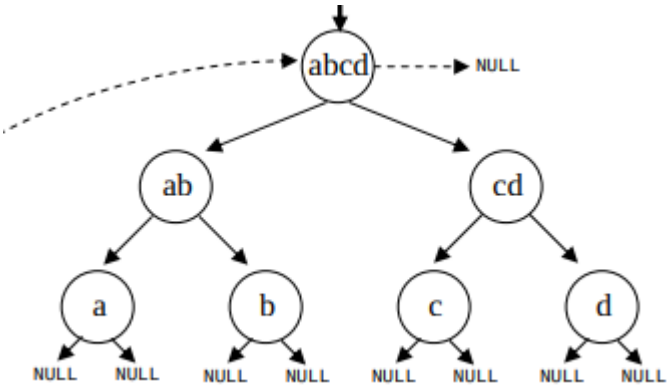

Fig. 5. Árboles de Huffman relacionados

Compresión de Huffman

La codificación de Huffman utiliza el concepto del ***mapping*** del alfabeto a una diferente representación compuesta por cadenas de tamaño variable, de modo que los símbolos con una alta frecuencia en el archivo (es decir, alta probabilidad de aparecer en el texto) posean una representación más pequeña en contraste con las de menor frecuencia. De esta manera, se construye una estructura (árbol binario) óptima con respecto a su distribución de caracteres.

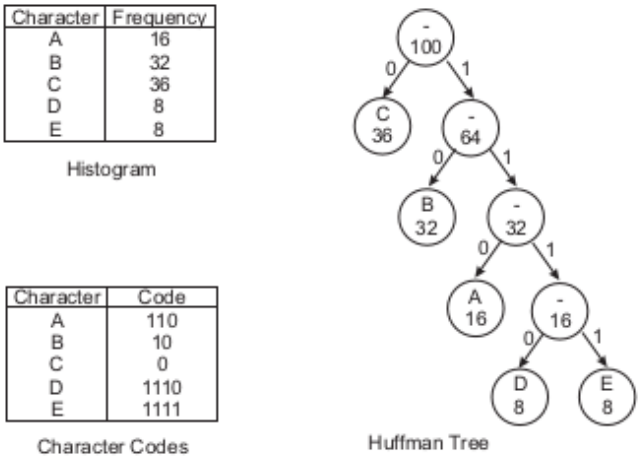

Fig. 6. Criterio de codificación de Huffman

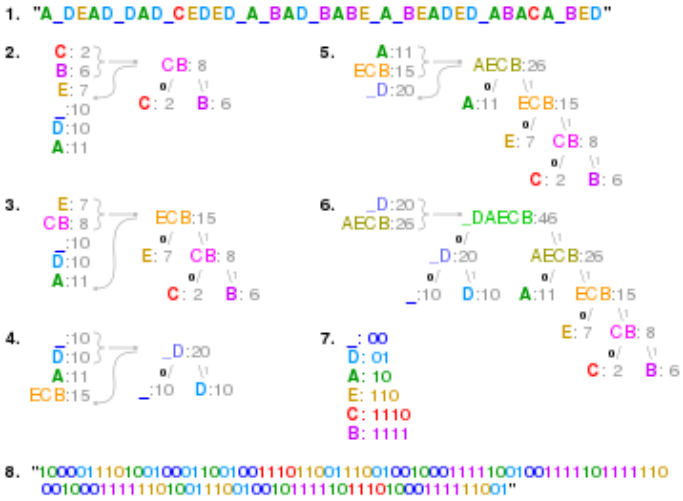

Fig. 7. Ejemplificación del algoritmo

La representación generada de cada símbolo resulta ser la más eficiente debido al importante criterio de establecer una codificación mínima a los símbolos de mayor ocurrencia.

Algoritmo

El algoritmo consiste en la construcción de un árbol binario cuyas hojas sean los símbolos presentes en el texto, de tal forma que al recorrer la estructura desde la raíz hasta cada una de sus hojas se obtenga el código Huffman asociado a cada carácter.

Para esto se realizan los siguientes pasos:

- Se crean árboles por cada uno de los símbolos del alfabeto del archivo, siendo cada uno de estos un nodo sin hijos, etiquetado con su carácter asociado y su frecuencia de ocurrencia.
- Se identifican los dos árboles de menor frecuencia y se unen creando un nuevo árbol. La etiqueta de la raíz será la suma de las frecuencias de las raíces de los dos árboles que se unen, y cada uno de estos árboles será un hijo del nuevo árbol. También se etiquetan las dos ramas del nuevo árbol: con un 0 la de la izquierda, y con un 1 la de la derecha.
- Se repite el paso 2 hasta que solo se encuentre un árbol que contenga a las demás.

Asumiendo la existencia de un archivo con el mensaje "**abracadabra**" y realizando la serie de pasos previamente definidos, la codificación Huffman de este sería el siguiente:

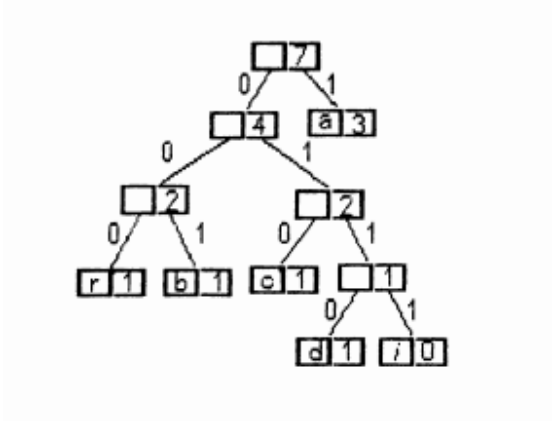

Fig. 8. Árbol binario resultante de cadena "abracadabra"

Puesto a que, en este ejemplo, existen caracteres con similar cantidad de ocurrencias en el archivo (frecuencias), se encuentran diversas maneras de estructurar el árbol de Huffman final. Esto implica una codificación por carácter distinta, dado a que esta se deriva del árbol resultante. Pese a esto, la eficiencia del algoritmo no se distorsiona. Un ejemplo de todo lo mencionado se evidencia en el contraste de la estructura del árbol resultante de la Fig. 8 y el de la Fig. 11.

Implementación Estática

Para poder visualizar de una manera real y analizar los cambios en los resultados, se ha implementado el código de compresión de Huffman en el lenguaje de programación *Python*.

Funcionamiento

Para poder simular una compresión de datos, el programa solicita al usuario acceder al documento a comprimir por medio de una interfaz gráfica. Una vez seleccionado el documento, se procede a almacenar todo el contenido de este en una variable de tipo $String$ y, seguidamente, a crear un archivo y escribir en este la traducción binaria del documento leído. Esto ayudará a determinar y visualizar la variación de pesos en los archivos reales. Luego, el programa comienza a ejecutar el algoritmo de Huffman explicado en la sección $Algoritmo$. El resultado del programa establece una ruta binaria por cada símbolo del archivo leído. Asimismo, muestra en pantalla los porcentajes relevantes de compresión. Finalmente, escribe el resultado codificado en un archivo de destino.

```
#Elaborado por Fabio López

def main():

    #La finalidad del algoritmo es reducir el tamaño en bits
    #de cada letra en un mensaje

    #La estrategia consiste en crear una ruta binaria por cada
    #letra presente en el mensaje, considerando la frecuencia de
    #esta en ella

    #Se creará un árbol binario, dado que en el último nivel de
    #este (mayor tiempo y dificultad de búsqueda) se encuentren
    #las letras de menor frecuencia, mientras que en los niveles
    #superiores (menor tiempo y dificultad de búsqueda) se encuentr
    #los de mayor frecuencia

    print("Compresión Huffman:")
    print("-------------------")
    print()

    ruta = '/home/fabio/Escritorio/Algoritmia/Huffman/PRUEBAHUFFMAN
    archivo = open(ruta,'r')
    dato = archivo.read()
    archivo.close()

    #generamos el archivo binario de ese documento para hacer una
    #simulación de comparación real
    archivob=open('/home/fabio/Escritorio/Algoritmia/Huffman/OUTPUT

    datobin=''
    for x in dato:
        datobin+=bin(ord(x))[2:].zfill(8)

    archivob.write(datobin)
```

Fig. 9. Implementación del algoritmo en Python

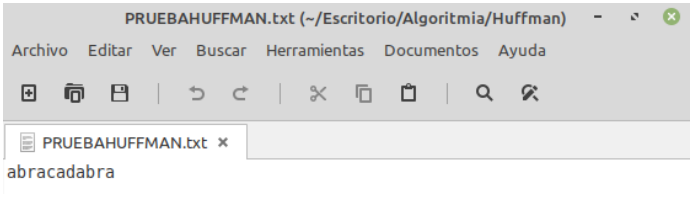

Fig. 10. Archivo de texto a comprimir

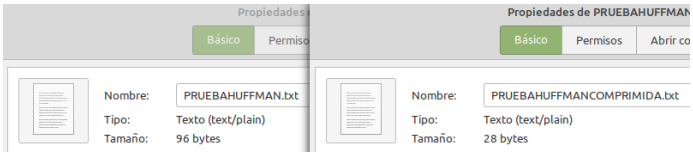

Fig. 11. Contraste de pesos entre el archivo original y su comprimido

Interpretación Matemática del Resultado:

Siendo la cadena original $S = \{'abracadabra(eof)'\}$ de alfabeto $A = \{'a','b','r','c','d','(eof)'\}$, la codificación final por carácter resulta ser:

$$C('a') = 0$$
$$C('b') = 111$$
$$C('r') = 110$$
$$C('c') = 1010$$
$$C('d') = 1011$$
$$C(eof) = 100$$

Y, por ende, la cadena original codificada por el algoritmo de Huffman resultaría:

$$C('abracadabra(eof)') = 011111001010010111011111100100$$

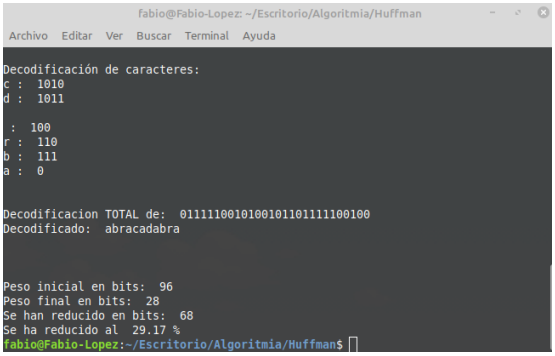

Fig. 12. Salida de programa elaborado

El resultado final de la compresión establece 28 bits para codificar los 12 caracteres originales. Por tanto, se han utilizado 28/12 = 2,33 $bits$ por símbolo.

Sustento Matemático

Caso Estático

La entropía es un concepto introducido que permite representar, matemática y aproximadamente, los límites de la codificación de los datos sin necesidad de conocer la naturaleza de los mismos. Esta representa una medida mínima, o cota inferior, de longitud de palabra por código para un alfabeto y sus respectivos pesos asociados.

Dado a que el cálculo de la entropía general no se puede calcular, se suele emplear la entropía de primer orden como una aproximación:

$$H = \sum_{a_i \in A} P(a_i) \log_2(\frac{1}{P(a_i)}) \quad (3)$$

- *P(a)*: Probabilidad del elemento.
- a_i: Elemento del alfabeto A.

donde,

$$P(a) = \frac{frecuencia}{l} \tag{4}$$

Utilizando el ejemplo anterior, la entropía resultaría:

$$H = P(a)\log_2(\frac{1}{P(a)}) + ... + P(eof)\log_2(\frac{1}{P(eof)})$$
$$H = \frac{5}{12}\log_2(\frac{12}{5}) + ... + \frac{1}{12}\log_2(\frac{12}{1})$$
$$H \approx 2,28$$

Finalmente, aunque la entropía explica que los datos se puedan codificar utilizando 2,28 bits por símbolo, el resultado final por Huffman expresa una codificación mayor de 2,33 bits por símbolo. La razón de este contraste radica en que la entropía solo indica un valor teórico mínimo (cota inferior) de los bits por símbolo necesarios, mientras que el resultado de la implementación del algoritmo de Huffman indica el valor real de bits por carácter del ejemplo.

Caso Dinámico

Los métodos estáticos de codificación presentan una rápida aplicación debido a que sus algoritmos son más sencillos (las probabilidades de ocurrencia de cada carácter ya están definidos). Sin embargo, en un método dinámico estas probabilidades se van calculando.

A continuación se detallan las definiciones necesarias para usar el método de Huffman aplicado al caso dinámico:

Definición 1: La condición necesaria para la codificación es la obtención de un código prefijo. Esto se puede implementar bajo la estructura de un árbol con algunas características especiales:

Sea $F = \{s_1, s_2, ..., s_q\}$ una fuente de caracteres, se define como árbol de codificación en base-r *Ar* si cumple:

- Ar es r-ario extendido
- Cada nodo interno es la suma de los pesos de sus hijos correspondientes
- Cada hoja se corresponde con uno y solo un símbolo y su peso respectivo

Definición 2: Sea $F = \{s_1, s_2, ..., s_q\}$ una fuente de caracteres, $P = \{p_1, p_2, ..., p_q\}$ las probabilidades de ocurrencia de cada carácter y $L = \{l_1, l_2, ..., l_q\}$ las longitudes de cada carácter en un código, se define la longitud promedio L en un código como:

$$L = \sum_{i=1}^{q} P_i L_i \tag{5}$$

Definición 3: Sea $F = \{s_1, s_2, ..., s_q\}$ una fuente de caracteres, $P = \{p_1, p_2, ..., p_q\}$ las probabilidades de ocurrencia de cada carácter y $L = \{l_1, l_2, ..., l_q\}$ las longitudes de cada carácter en un código, se define la varianza V de un código como:

$$V = \sum_{i=1}^{q} P_i (L_i - L)^2 \tag{6}$$

Definición 4: Sea $C = \{x_1, x_2, ..., x_q\}$ un código para una fuente *F*, se dice que es ordenado si cumple: Sean S_i y S_j, dos símbolos de la fuente, con P_i y P_j; sus correspondientes probabilidades de ocurrencia, X_i y X_j las palabras código; para todo i, j, tal que $P_i > P_j : X_i < X_j$

Definición 5: Sea A, un árbol Huffman, y sea n la cantidad de nodos internos de A, la cantidad de nodos de A se define como de 2^n.

Definición 6: Sea A, un árbol huffman, y sea $F = \{s_1, s_2, ..., s_q\}$ una fuente, la cantidad de nodos de A, se define como de $2^{(q-l)}$.

CONCLUSIÓN

El algoritmo estudiado en el presente informe ($Huffman$) ofrece una manera de comprimir información sin pérdidas al trabajar con archivos de texto simple. Para su correcto funcionamiento, se debe conocer inicialmente las frecuencias de cada carácter en el archivo. Estas frecuencias serán usadas para generar el árbol binario que, finalmente, establecerá las claves de cada símbolo respecto a su recorrido desde la raíz.

El determinante clave en el proceso de comprensión será el número de veces que se repitan los caracteres en el archivo. De esta manera, y explicado por el concepto de $Entropía$, a mayor número de repetición mejor funcionamiento del algoritmo. Contrariamente, a un menor número de repeticiones, la comprensión no dará resultados demasiado satisfactorios.

REFERENCES

[1] Rueda, L. G., Ortega, M. O., Klenzi, R. O., Rodríguez, N. R. (1995). Un modelo de codificación dinámica del método de Huffman para la compresión de datos en línea. In I Congreso Argentino de Ciencias de la Computación.

[2] Sharma, M. (2010). Compresión con codificación Huffman. IJCSNS International Journal of Computer Science and Network Security , 10 (5), 133-141.

[3] Rigler, S., Bishop, W., Kennings, A. (2007, April). FPGA-based lossless data compression using Huffman and LZ77 algorithms. In 2007 Canadian conference on electrical and computer engineering (pp. 1235-1238). IEEE.

[4] Sangwan, N. (2012). Text encryption with huffman compression. International Journal of Computer Applications, 54(6).

[5] Lopez, J. (2010). Algoritmo de Huffman. joselu.webs.uvigo.es, 1-3. http://joselu.webs.uvigo.es/material/Algoritmo

[6] Introducción a la compresión: Huffman y Entropía. Universidad Politécnica de Valencia. Prácticas Txón Datos Multimedia. Valencia, España: Grupo de Redes de Computadores. http://www.grc.upv.es/docencia/tdm/practicas/P1.pdf