[[1]](#footnote-1)

INFORME DE LABORATORIO #2

CODIGO HUFFMAN

María Camila González, Alejandra Niño, Juan Sebastián Silva.

# ***Abstract***— This laboratory consisted of performing the software representation of the HUFFAM source code.

# ***Key words***— software, MATLAB, HUFFMAN code, SOURCE coding.

# ***Resumen***— Este laboratorio consistió en realizar la representación en software de la codificación de fuente HUFFAM.

# ***Palabras clave***—MATLAB, codificación de fuente, Código HUFFMAN,

# **INTRODUCCIÓN**

# En el sector de las telecomunicaciones se tiene una tarea importante a realizar, la cual es, el análisis y diseño de enlaces de comunicaciones digitales y para esto se requiere de un aspecto importante el cuál es la transmisión, para esto se necesita hacer codificaciones, tanto de línea, fuente y canal, el segundo .se utiliza para la compresión de datos, el cual se analizará en este documento.

# Es por esto que en el presente documento se presentarán los resultados de la práctica de laboratorio en que se realizó en la materia de sistemas y telecomunicaciones 2, el cuál consistía en realizar en un lenguaje de programación el código Huffman, el cual es usado para la compresión de datos, donde se codifica un símbolo, como un carácter de un mensaje basándose en la probabilidad de aparición de cada carácter del abecedario proveniente del mensaje.

# **MARCO TEÓRICO**

Java:  es un [lenguaje de programación](https://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n) de [propósito general](https://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n_de_prop%C3%B3sito_general) que se deriva de [C++](https://es.wikipedia.org/wiki/C%2B%2B)., además este lenguaje está [orientado a objetos](https://es.wikipedia.org/wiki/Programaci%C3%B3n_orientada_a_objetos) que fue diseñado para tener tan pocas dependencias de implementación[1].

Codificación: es el proceso en el que se realiza una conversión de símbolos para que una información pueda ser comunicada pueda ser entendida por el receptor [2].

Compresión de datos: Es el reducir el tamaño físico de la información. Esto se realiza mediante algún algoritmo pueda utilizar los datos al teniendo las consideraciones de cada tipo de datos que se van a comprimir[3].

Codificación de fuente: Esta codificación consiste en la asignación de códigos binarios a información textual[4].

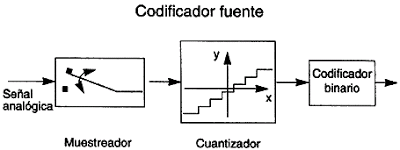


Fig2. Codificador de fuente [5].

Código Huffman: un método estadístico en el que se le asigna un código binario a los símbolos que se van comprimir. La longitud de cada código es distinta para todos los símbolos ya que a los códigos cortos son para los símbolos más usados y para los códigos largos son para los símbolos menos frecuentes [6].

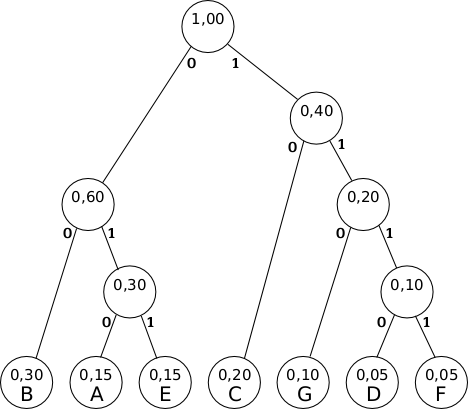


Fig3. Ejemplo código Huffman [7].

# **PROCEDIMIENTO**

En este laboratorio se procedió a realizar la codificación Huffman en un lenguaje de programación en este caso, Java con una interfaz GUI. Para realizar este procedimiento se realizó separación de procesos para unir todo al final.

Se tiene la siguiente interfaz gráfica:

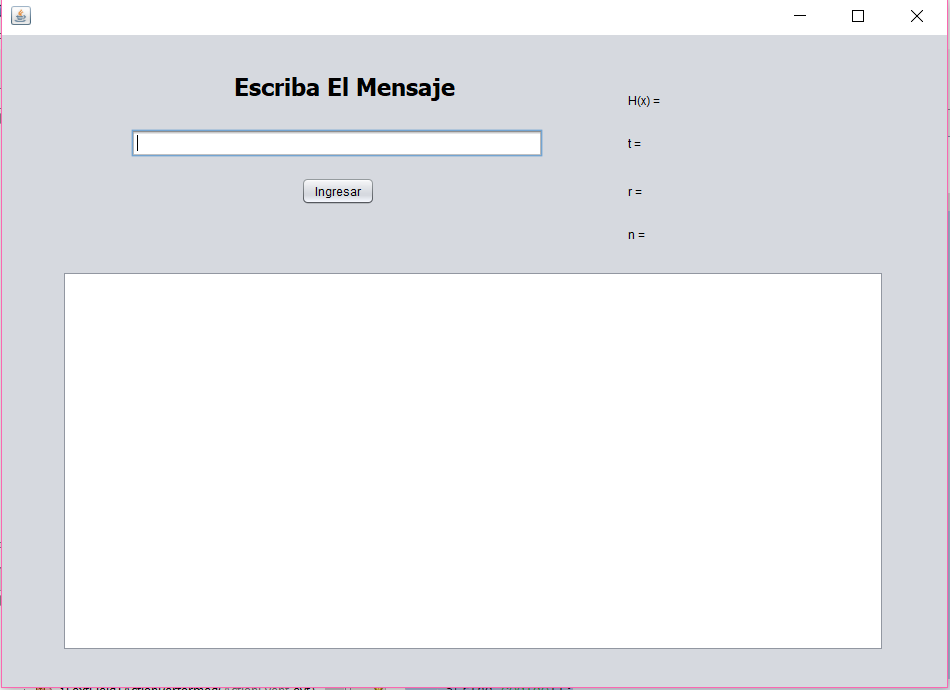


Fig 4. Interfaz Gráfica de Usuario

Aquí el usuario deberá ingresar el mensaje no mayor a 35 caracteres.

1. Primero se crean las **variables**:

public String Mensaje = "";

public String Total = "";

public char caracteres[];

public char alfabeto[] = new char[35];

public char alfabeto2[] = new char[35];

public double contadores[] = new double[35];

public double contadores2[] = new double[35];

public int cont = -1;

public char temporal;

public char temporal12;

public double temporal2;

public double temporal4;

public double temporal5 = 0;

public int temporal3;

public double temporal22;

public double temporal52 = 0;

public int temporal32;

public int saber = 0;

public int locacion0=0;

public int locacion1=0;

public double total = 0;

public double parcial = 0;

public double div[] = new double[35];

public double div2[] = new double[35];

public double tam = 0;

public int posi=0;

String codigo[];

1. Luego se **obtiene el mensaje** ingresado por el usuario:

this.jTextArea1.setEditable(false);

Mensaje = this.jTextField1.getText();//Obtiene el mensaje que ingreso el usuario

caracteres = this.Mensaje.toCharArray();//Convierte el mensaje en un arreglo de caracteres

1. Tercero se analiza el mensaje, mirando donde hay espacios y se construye el **alfabeto** del mensaje:

tam = caracteres.length;//obtiene el tamaño del arreglo de caracteres

cont = -1;//Variable que recorre el alfabeto

if (tam <= 35) {//Restringe que el mensaje no tenga mas de 35 caracteres

for (int i = 0; i < caracteres.length; i++) {//Ciclo que recorre el arreglo de caracteres

if (caracteres[i] == ' ') {//Mira si en esa posicion hay un espacio

caracteres[i] = '\_';//reemplaza esos espacios por \_

}

}

for (int i = 0; i < caracteres.length; i++) { //Ciclo que recorre el arreglo de caracteres para identificar el alfabeto

temporal = caracteres[i];//asigna a la variable temporable el caracter a analizar

for (int j = 0; j < caracteres.length; j++) {//ciclo que recorre el arreglo de alfabeto

if (temporal == alfabeto[j]) {//mira si la variable a analizar se encuentra en el arreglo de alfabeto

saber = 1;//si está en alfabeto le asigna a saber 1, porque estaría repetida

}

}

if (saber == 0) {//mira si el carácter no está repetido

cont++;//aumenta para asignar en esa posición de alfabeto

alfabeto[cont] = temporal;//en la posición que tiene cont en el arreglo alfabeto guarda la variable que se analizo

} else {//si el carácter esta repetido

saber = 0;//se reinicia la variable saber para analizar otro caracter

}

}

for (int i = 0; i < caracteres.length; i++) {//Ciclo que recorre el arreglo de alfabeto para saber cuántas veces se ha repetido un caracter

for (int j = 0; j < caracteres.length; j++) {//Ciclo que recorre el arreglo de caracteres para saber cuantas veces se ha repetido un caracter

if (alfabeto[i] != ' ') {//mira si en la posición de alfabeto no esta " "

if (alfabeto[i] == caracteres[j]) {//mira si el caracter a analizar se encuentra en el arreglo caracteres

contadores[i] = contadores[i] + 1;//cuenta cuantas veces esta

}

}

}

}

1. Luego de esto se halla la **probabilidad** de cada carácter del alfabeto de la siguiente manera:

for (int i = 0; i < caracteres.length; i++) {//Ciclo que recorre el arreglo de caracteres para hallar la probabilidad

if (contadores[i] != 0) {//mira si lo que esta en esa posición es diferente de 0

div[i] = contadores[i] / tam;//halla la probabilidad de cada caracter y lo guarda en el arreglo div

parcial = div[i] \* ((Math.log(div[i])) / Math.log(2));//halla la cantidad de información de cada caracter

total = total - (parcial);//suma las cantidades de información

}

}

System.out.println(total); //imprime la entropia por consola

this.jLabel1.setText("H(x) = "+total+"."); //Modifica el jLabel1 con la entropia

1. Luego se **ordenan** las probabilidades de mayor a menor con la función ordenar();

public void ordenar() {

double conta=0;

for (int i = 0; i < caracteres.length; i++) { //Ciclo que va desde 0 hasta la longitud del mensaje

for (int j = 0; j < caracteres.length; j++) { //Ciclo que va desde 0 hasta la longitud del mensaje

temporal4 = div[j]; // Guarda la probabilidad a analizar

if (temporal4 > temporal5) { //Mira si es mayor a un punto de referencia que se tiene

temporal5 = temporal4; //Si es mayor se asigna como nuevo punto de referencia

temporal3 = j; // guarda la posición en la que se encuentra

temporal2 = div[j]; // guarda la probabilidad

temporal = alfabeto[j]; // guarda el caracter que se esta analizando

conta=contadores[j]; //guarda la cantidad de veces que esta ese contador

}

}

contadores2[i]=conta; // guarda la ultima cantidad de caracter que se analizo

div2[i] = temporal2; //guarda en div2 la probabilidad

alfabeto2[i] = temporal; // guarda en alfabeto2 el caracter que se analizo

div[temporal3] = 0; //guarda en div en la posición que se analizo por ultima vez un 0

temporal5 = 0;//se reinicia temporal5

}

for (int i = 0; i < caracteres.length; i++) { //Ciclo que va desde 0 hasta la longitud del mensaje

div[i] = div2[i]; //guarda en div lo que esta en div2

contadores[i]=contadores2[i]; //guarda en contadores lo que esta en contadores2

}

}

1. En esta parte se realizan las **sumas** necesarias en la codificación Huffman, empezando por los números con menores probabilidades y así sucesivamente hasta llegar a 1:

codigo = new String[posi];//crea un arreglo llamado codigo de tamaño posi

int orden[] = new int[posi];//crea un arreglo de enteros llamado codigo de tamaño posi

int saber = 0;//crea variable para saber si ya tomo un o dos numero para sumar

int saber2= 0;//crea variable para saber si ya se sumaron

int revisar = 0;//Saber cuando salir del ciclo

for(int i=0; i<posi;i++){//ciclo para recorrer codigo y orden

orden[i]=i;//asigna valores de 0 hasta n para el orden

codigo[i]="";//crea vacios en el arreglo codigo

}

while (saber != 5) {//inicia ciclo para hacer las sumas

temporal5 = 10;//Comparar si hay mayores o menores iniciando con un valor grande

temporal52 = 10;//Comparar si hay mayores o menores iniciando con un valor grande

for (int j = posi - 1; j >= 0; j--) { //ciclo que recorre div para encontrar el menor

temporal4 = div2[j];//temporal4 guarda el valor que se esta analizando

if(saber2==0){ //mira el estado de saber 2 para el primer numero menor

if (temporal4 < temporal5 && temporal4 != 0 ) { //mira si temporal4 es menor a temporal5 y si temporal4 es diferente a 0

temporal5 = temporal4;//se le asigna a temporal5 lo que esta en temporal4

temporal3 = j;// guarda la posicion en la que esta el numero menor

temporal2 = div2[j]; //guarda el valor de la menor probabilidad

}

}

if(saber2==1){ //mira el estado de saber 2 para el segundo numero menor

if (temporal4 < temporal52 && temporal4 != 0 ) { //mira si temporal4 es menor a temporal5 y si temporal4 es diferente a 0

temporal52 = temporal4; //se le asigna a temporal5 lo que está en temporal4

temporal32 = j; // guarda la posición en la que está el número menor

temporal22 = div2[j]; //guarda el valor de la menor probabilidad

}

}

if(j==0 && saber2==0){ // antes de salir del for dice cuál es el número menor y lo vuelve 0

div2[temporal3] = 0; // le asigna en esa posicion un 0

saber2++; // aumenta saber2 para buscar el siguiente número menor

}

else if(j==0 && saber2==1){ // antes de salir del for dice cuál es el segundo número menor y lo vuelve 0

div2[temporal32] = 0; // le asigna en esa posición un 0

saber=1; // para saber que ya encontró los dos números menores

}

// System.out.println(div2[j]);

}

// System.out.println();

if (saber == 1) { //mira si ya encontró los dos números menores

div2[temporal3] = temporal2 + temporal22;//suma los dos números menores

if(temporal22>=temporal2){ // mira si temporal22 es mayor o igual a temporal2

locacion0=orden[temporal32]; // le asigna el valor de orden en la posición temporal32 para asignarle 0

locacion1=orden[temporal3]; // le asigna el valor de orden en la posición temporal13 para asignarle 1

for(int i=0;i<posi;i++){ //ciclo que va desde 0 hasta posición

if(orden[i]==locacion0){ // mira si orden en la posición i es igual a lo que guardo locacion0 hará:

codigo[i]=codigo[i]+"0"; //a código en esa posición se le asigna un 0

orden[i]=locacion0; // orden en la posicion i recibe el valor de locacion0

}

if(orden[i]==locacion1){ // mira si orden en la posición i es igual a lo que guardo locacion1 hará:

codigo[i]=codigo[i]+"1"; ////a código en esa posición se le asigna un 1

orden[i]=locacion0; // orden en la posición i recibe el valor de locacion0

}

}

}

revisar = 0; // reinicia la variable revisar

saber = 0; // reinicia la variable saber

saber2 =0; //reinicia la variable saber2

for (int i = 0; i < posi; i++) { //ciclo que va de 0 hasta posición

if (div2[i] != 0) { // mira si div2 en la posición i es diferente a 0

revisar++; // aumenta revisar

}

}

if (revisar == 1) { // mira si solo hay un número diferente de 0

saber = 5; // si solo hay un número diferente de 0 le asigna a saber 5 para salirse

}

}

}

1. Como últimos pasos se hace **llamado de cuatro funciones**:
2. **revertir();** // llama la función revertir la cual ordena los códigos de cada letra del alfabeto:

public void revertir(){

for(int i=0;i<posi;i++){ //Ciclo que va desde 0 hasta posi

char tempo[]=codigo[i].toCharArray(); //crea un arreglo de char proveniente de lo que se encuentra en codigo en esa posición

int longitud=tempo.length; //guarda la longitud del anterior arreglo

char tempo2[]=new char [longitud]; //crea un arreglo tipo char de tamaño longitud

int k=longitud-1;//se inicia una variable que va desde la ultima posición

for(int j=0;j<longitud;j++){//Ciclo que va desde 0 hasta longitud

tempo2[j]=tempo[k];//guarda en tempo2 lo que esta en tempo desde la posición k

k--;//k ira disminuyendo para guardar el codigo de forma contraria

}

codigo[i]="";//se asigna "" a cada posición de codigo

for(int a=0;a<longitud;a++){//Ciclo que va desde 0 hasta longitud

codigo[i]=codigo[i]+tempo2[a];//se le asigna a cada posición de codigo lo que esta en codigo mas lo que esta en tempo2

}

}

}

1. **longitudprom();** // llama la función longitudprom, que calcula la longitud promedio de la codificación:

public void longitudprom(){

double sumas=0;

char numbits[]; // crea un arreglo de chars

for(int i=0;i<posi;i++){ //ciclo que va de 0 a posición

numbits=codigo[i].toCharArray(); // se le asigna numbits el arreglo de chars lo que tenga el arreglo código en la posición i

sumas=sumas+(numbits.length\*div[i]); // suma los valores de la multiplicación por la cantidad de bits del código

}

longiprom=sumas; // se le asigna a longitudprom el resultado de sumas

this.jLabel2.setText("l = "+sumas+"."); // modifica jlabel2 con el valor de sumas

}

1. **tasacompresion();** // llama la función tasacompresion, que calcula la tasa de compresión de la codificación:

public void tasacompresion(){

double sumas=0;

sumas=((Math.log(posi)) / Math.log(2))/longiprom; //calcula la tasa de compresión

this.jLabel3.setText("r = "+sumas+"."); //modifica el jlabel3 con el valor de la tasa de compresión

}

1. **eficacia();** // llama la función eficacia, la cual calcula la eficacia de la codificación Huffman:

public void eficacia(){

double sumas=0;

sumas=total/longiprom; // calcula el valor de la eficacia

this.jLabel4.setText("n = "+sumas+"."); //modifica el jlabel4 con el valor de la tasa de la eficacia

}

1. Por último, se obtienen los siguientes **resultados** con el ejemplo de: “la clase de teleco2 es interesante”:

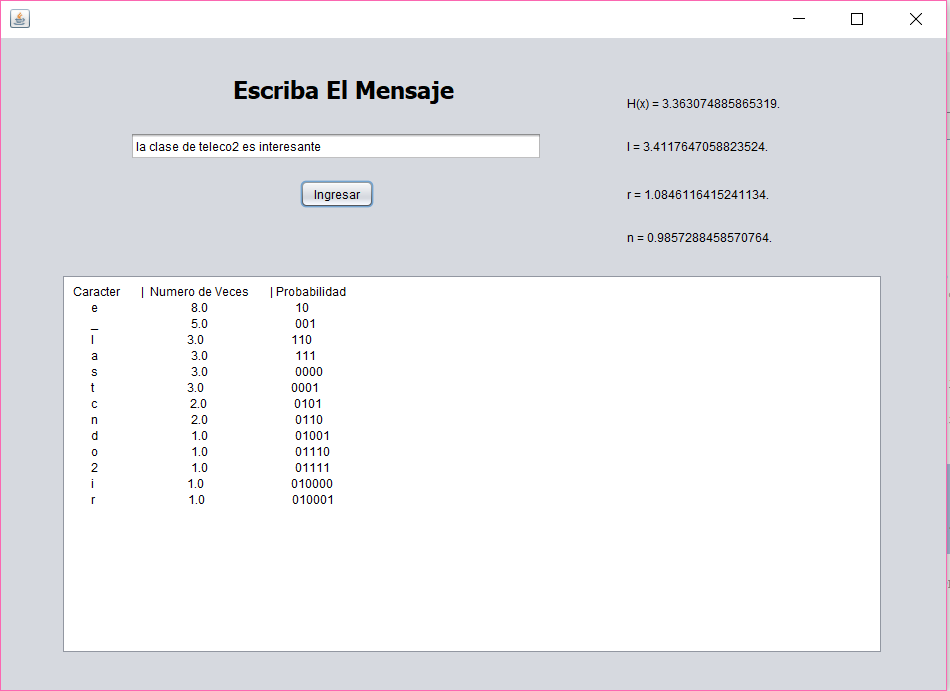


Fig 5. Resultados obtenidos

Se obtiene una codificación huffman con criterios de orden de aparición y ubicación a la derecha donde la longitud promedio que es de 3.411 bits por símbolo se asemeja lo que se espera en la entropía y con una tasa de compresión de 0.86 para este ejemplo.

# **CONCLUSIONES**

1. Con la anterior practica de laboratorio se pudo concluir que, la codificación de fuente Huffman para la compresión de datos para la transmisión de estos se puede realizar mediante diferentes softwares de programación, como Java, MATLAB y demás, pero siendo este proceso bastante difícil de codificar, que puede facilitarse mediante funciones ya preestablecidas de estos softwares, como en el caso de MATLAB que no se encuentran en Java.
2. La codificación Huffman puede variar en el orden en que codifica cada carácter dependiendo de el orden de análisis, pero esto no indica que esté erróneo, solo que se hizo bajo cierto parámetro.
3. Para desarrollar este algoritmo no es necesario estar reorganizando el arreglo de probabilidades sino analizarlo para saber dónde están las menores probabilidades.
4. Los resultados obtenidos en este tipo de software pueden variar con los hechos a mano debido a que la memoria de estos dispositivos le permite manejar más decimales, en donde a mano sería algo engorroso.
5. Existen varios métodos para codificar este tipo de algoritmos, dependiendo la forma en que se deseen procesar los datos.

# **REFERENCIAS**

[1] *Programming Language Popularity*. 2009.

[2] «Significado de Codificación - Qué es, Definición y Concepto». [En línea]. Disponible en: http://quesignificado.com/codificacion/. [Accedido: 28-ago-2017].

[3] «La compresión de datos». [En línea]. Disponible en: http://es.ccm.net/contents/714-la-compresion-de-datos. [Accedido: 12-sep-2017].

[4] «Codificacion de fuente y de canal de andres builes en Prezi». [En línea]. Disponible en: https://prezi.com/mdor2e8mxcwc/codificacion-de-fuente-y-de-canal/. [Accedido: 12-sep-2017].

[5] «Clase 2». [En línea]. Disponible en: http://clase2teleco.blogspot.com.co/. [Accedido: 12-sep-2017].

[6] «Código Huffman». [En línea]. Disponible en: http://es.ccm.net/contents/728-codigo-huffman. [Accedido: 12-sep-2017].

[7] «File:ArbolCodigoHuffman.png - Wikimedia Commons». [En línea]. Disponible en: https://commons.wikimedia.org/wiki/File:ArbolCodigoHuffman.png. [Accedido: 12-sep-2017].

1. [↑](#footnote-ref-1)