




TEORÍA DE LA  
INFORMACIÓN

# TRABAJO INTEGRADOR N° 1



GRUPO 4 - Integrantes:

Reale, Valentina - [valenreale123@gmail.com](mailto:valenreale123@gmail.com)

Rodriguez, Juan Gabriel - [nogren23@gmail.com](mailto:nogren23@gmail.com)

Presa, Martiniano - [martiniano.presaaa@gmail.com](mailto:martiniano.presaaa@gmail.com)

---

# ÍNDICE

<b>Resumen</b>	<b>2</b>
<b>Introducción</b>	<b>2</b>
<b>Primera Parte</b>	<b>3</b>
Enunciado	3
Resolución	3
Extensión de orden 20	4
<b>Segunda Parte</b>	<b>5</b>
Enunciado	5
Resolución	5
Lectura de archivo	5
Cálculo entropía y cantidad de información	6
Códigos obtenidos	7
Inecuación de Kraft, MacMillan	8
Longitud Media	9
Rendimiento y redundancia	9
Codificación Huffman	10

---

# Resumen

El presente trabajo examina una fuente de información con tres símbolos diferentes (A,B,C) con un total de 10.000 símbolos. Para estudiar una fuente de estas propiedades es necesario construir un programa que realice los cálculos necesarios para realizar un análisis adecuado. Este trabajo contiene información sobre cómo se pensaron, analizaron y fueron desarrollados los códigos más importantes con el objetivo de recopilar información para luego realizar los estudios correspondientes.

Los estudios realizados en el informe se refieren tanto a las características y comportamiento de la fuente proporcionada como a la aplicación de recursos teóricos a los datos recogidos de la fuente.

# Introducción

El trabajo se basa en analizar la fuente de información propuesta tomando palabras código de diferente número de dígitos para más tarde, con la ayuda de un programa hecho específicamente para esta tarea el cual se encontrará para su debida visualización en en siguiente [enlace](#), se estudie las características de la fuente, teniendo en cuenta factores como su cantidad de información por símbolo, su entropía, si es una fuente de memoria nula, etc.

---

# Primera Parte

## Enunciado

A partir de un archivo provisto por la cátedra que contiene 10.000 caracteres aleatorios de un alfabeto {A, B, C}:

- Calcular las probabilidades condicionales (que un símbolo se dé, si se dio otro) y en base a ello determinar si es una fuente de memoria nula o no nula.
- En caso de ser una fuente de memoria nula, generar la extensión de orden 20 y calcular la entropía de la fuente inicial y la de orden 20.
- En caso de ser una fuente de memoria no nula, determinar si es ergódica y en caso de serlo establecer su vector estacionario. Calcular la entropía de la fuente.

## Resolución

A partir del archivo que contiene 10.000 caracteres aleatorios de un alfabeto {A, B, C} calculamos la frecuencia de aparición de cada carácter. De esta manera, con dicha frecuencia calculamos la probabilidad condicional de cada símbolo.

Matriz de apariciones condicionadas:

	A	B	C
A	135	368	586
B	383	1245	1877
C	571	1892	2942

Matriz de probabilidades:

	A	B	C
A	0,123967	0,104993	0,108418
B	0,351699	0,355207	0,347271
C	0,524334	0,539800	0,544311

Con la matriz de probabilidades condicionales, vemos que las probabilidades de un suceso habiendo ocurrido otro son muy similares para el mismo carácter, es decir, analizamos en cada fila cada carácter individualmente, de esta manera corroboramos la proximidad de ocurrencia del

---

caracter analizado, teniendo en cuenta la previa aparición de otro perteneciente al alfabeto código (A,B,C).

Por lo que tomando como criterio una cota de probabilidad de 0,02 por fila (que nos asegura una buena precisión en la mayoría de los casos) respecto de la probabilidad media del símbolo, podemos concluir que todas las probabilidades están dentro del intervalo, por lo que podemos concluir que la fuente es de memoria nula.

## Extensión de orden 20

Dado que la fuente es de memoria nula, podemos generar la extensión de la fuente de orden 20 y calcular su entropía.

La cantidad media de información la calculamos con un algoritmo que implementa la siguiente ecuación:

$$H(S) = \sum_S P(S_i) \log \frac{1}{P(S_i)}$$

resultando la entropía de la fuente inicial:

**Entropía fuente: 0.8569696552134656**

La misma propiedad se aplica para la fuente de orden 20:

$$H(S^n) = \sum_{S^n} P(\sigma_i) \log \frac{1}{P(\sigma_i)}$$

De esa manera calculamos directamente la entropía de la extensión de la fuente. Dejamos correr el programa más de 40 minutos y aún no finalizaba el cálculo debido a que existen 3.486.784.401 posibles combinaciones (símbolos diferentes) y la operación resulta muy larga. Nos dimos cuenta que se podía calcular de una manera mucho más sencilla: a partir de la entropía ya calculada y de la siguiente propiedad:

$$H(S^n) = n H(S)$$

Siendo n = 20 y H(S) la entropía recién calculada, resulta:

**Entropía orden 20: 17.13939310426931**

---

# Segunda Parte

## Enunciado

Tomando el archivo precedente, considerar que las cadenas representan palabras de un código:

- de 3 caracteres.
- de 5 caracteres.
- de 7 caracteres.

Identificar en cada caso las palabras código y en base a ellas y su frecuencia:

- a) Calcular la cantidad de información y entropía.
- b) Los códigos que se obtuvieron, ¿de qué tipo son? Justificar su respuesta.
- c) Establecer en cada caso la Inecuación de Kraft, MacMillan, Longitud Media del código y si cumplen con la condición de ser compactos. Obtener conclusiones.
- d) Determinar el rendimiento y redundancia de cada código. Obtener conclusiones.
- e) Codificar los símbolos de los códigos anteriores según Huffman o Shannon-Fano (a elección) y reconstruir el archivo (en tres archivos, uno por codificación). Obtener conclusiones.

## Resolución

Para calcular la cantidad de información, se debe realizar el logaritmo de 1 sobre la probabilidad del símbolo. Para calcular la entropía, se realiza la sumatoria de las probabilidades de cada símbolo multiplicadas por la cantidad de información del mismo. Para poder llevar a cabo estos cálculos diseñamos una solución la cual lee, del archivo de texto propuesto, la cantidad de caracteres que se necesiten para formar una palabra.

## Lectura de archivo

Para separar el texto en términos de  $n$  caracteres, primero leímos todo el archivo y guardamos en una variable de tipo String 'mensaje', el texto leído. Luego, a partir del método `String.substring()`, extraemos del String que se utiliza para invocarlo (en este caso 'mensaje') subcadenas de texto dando las posiciones iniciales y finales. Cada una de las subcadenas se guarda en un HashMap de forma tal:

MAP <String, Register> código
-------------------------------

---

Siendo la clave de tipo String el símbolo correspondiente al del objeto Register, una clase que contiene cada símbolo de n caracteres junto a su frecuencia de aparición. La ventaja de utilizar una estructura como tal es que al leer cada símbolo nuevo del texto, se verifica fácilmente si ya se encuentra en la misma; en ese caso solo se aumenta la frecuencia en 1, en caso contrario, se agrega el símbolo al HashMap con frecuencia 1.

Además de eso, tenemos una estructura de tipo ArrayList <String>:

`ARRAYLIST <String> indice`

donde guardamos cada símbolo diferente para luego usarlo como índice al buscar en el HashMap, ya que no permite recorrido iterativo, sino que solo se obtiene el objeto de tipo Register a partir de la clave String.

## Cálculo entropía y cantidad de información

Una vez finalizada la lectura del archivo y cargadas las estructuras ya nombradas, estamos en condiciones de calcular la cantidad de información de cada uno de los símbolos y su entropía:

```
Frecuencia total := 0

para ( i=0 hasta Tamaño del Array Indice con pasos de 1 ){
    Frecuencia total := Frecuencia total + Frecuencia del Índice
    i
}

para (i=0 hasta Tamaño del Array Indice con pasos de 1){
    Probabilidad en Map Código := Frecuencia del Índice i /
Frecuencia total
    Log := Logaritmo en base 3 de ( 1 /probabilidad del Índice i
)
    Cantidad de Información = Cantidad de Información + Log
    Entropía = Entropía + probabilidad del Índice i * Log
}
```

---

En el primer escenario, con palabras de 3 caracteres, se obtienen 28 símbolos diferentes y resultan:

$$\text{Entropía} = 2.57$$

$$\text{CantInfo} = 102.9$$

En el segundo escenario, con palabras de 5 caracteres, se obtienen 199 símbolos diferentes y resultan:

$$\text{Entropía} = 4.23$$

$$\text{CantInfo} = 1084.01$$

En el tercer escenario, con palabras de 7 caracteres, se obtienen 624 símbolos diferentes y resultan:

$$\text{Entropía} = 5.54$$

$$\text{CantInfo} = 3822$$

#### Conclusiones:

Analizando los resultados obtenidos, notamos que la cantidad de información de aparición de cada palabra código aumenta conforme incrementamos la cantidad de dígitos, esto se produce ya que la cantidad de caracteres en la fuente no cambia, por lo tanto al incrementar la cantidad de dígitos cada palabra aporta mayor cantidad de información, al poseer una menor frecuencia dentro de la fuente, por lo tanto es menos predecible su comportamiento. Además, mientras mayor sea la cantidad de símbolos obtenidos, mayor será la información que nos aportan, por la misma razón.

### **Códigos obtenidos**

Todos los códigos propuestos (ya sean los de 3, 5 o 7 caracteres) son no singulares, debido a que ninguna de sus claves se repite, todas las palabras son diferentes entre sí en cada uno de los tres casos.

También son unívocamente decodificables, ya que no son singulares en su extensión de orden  $n$ , es decir que es imposible obtener mensajes distintos con una misma serie de palabras, cualquiera sea su extensión.

Con respecto a la instantaneidad, podemos afirmar que el código de palabras de 5 caracteres es instantáneo ya que ninguna palabra código es prefijo de otra. Esto es fácil de notar porque todas tienen la misma cantidad de dígitos y cumplen con la no singularidad. Por otro lado, en el caso de longitudes de 3 y 7 caracteres, resulta que el último símbolo tiene longitud de 1 y 4 caracteres respectivamente, por lo que sin lugar a duda resultan prefijos de algún otro símbolo perteneciente a su código, condición suficiente para que no sean instantáneos.



---

## Inecuación de Kraft, MacMillan

La inecuación de Kraft es la misma que la de MacMillan, la divergencia recae en que Kraft la definió como condición suficiente para la existencia de un código instantáneo, mientras que MacMillan la definió como condición necesaria.

El procedimiento que utilizamos para calcularla fue el siguiente:

```
ALGORITMO Kraft-McMillan;
VAR
  MAP codigo;
  ARRAYLIST indice;
  ENTERO longitud, r = 3; //r = nro de símbolos diferentes del alf
codigo
  DOUBLE kraft = 0;
INICIO
  Para (entero i=0; i<cant de símbolos diferentes; i++){
    longitud = longitud de cada símbolo;
    kraft += r ^ -longitud;
  }
  Escribir( kraft );
FIN
```

A partir de su resultado y sabiendo que si la inecuación de Kraft resulta  $\leq 1$  podemos afirmar que puede ser un código instantáneo (para asegurarlo hay que fijarse en la condición de prefijo) podemos concluir lo siguiente:

- En el primer escenario la inecuación de Kraft resulta 1.33, afirmando que es un código NO instantáneo. Además que sabemos que el último símbolo tiene longitud 1 y es “C”, siendo prefijo de varios de los símbolos restantes.
- En el segundo escenario la inecuación de Kraft resulta 0.819, y como tienen todos los símbolos igual longitud y son no singulares, podemos deducir que ninguno de ellos será prefijo de otro, por lo tanto, es un código instantáneo.
- En el tercer escenario inecuación de Kraft resulta 0.2972, podría ser instantáneo pero nuevamente el último símbolo “BBCC” es más corto que el resto que tiene 7 caracteres y resulta prefijo de otros.

---

## Longitud Media

Para calcular la longitud media del código en cada escenario desarrollamos el siguiente método:

```
FUNCION LongitudMedia;
VAR
    MAP codigo;
    ARRAYLIST indice;
    ENTERO longitud,
    DOUBLE probabilidad, longMedia;
INICIO
    Para (entero i=0; i<cant de símbolos diferentes del código; i++){
        longitud = longitud de cada símbolo;
        probabilidad = probabilidad de aparicion de cada simbolo
        longMedia += longitud * probabilidad;
    }
    Retorna( longMedia );
FIN
```

Dando como resultado:

Escenario 1: longMedia = 2.9994

Escenario 2: longMedia = 5

Escenario 3: longMedia = 6.9979

Por lo tanto podemos observar que la longitud media de cada código corresponde a sus longitudes reales por cada escenario. Esto quiere decir que los códigos de los tres casos presentes cumplen con la condición de ser compactos, ya que la longitud media es igual a la longitud de cada código.

Tanto la definición de longitud media como la de códigos compactos hacen referencia a las longitudes de los códigos y no a los códigos mismos, por lo tanto llegamos a la conclusión de que en todos los escenarios de dígitos posibles también se darían los mismos casos.

## Rendimiento y redundancia

Para calcular el rendimiento dividimos la entropía de la fuente por su longitud media, y la redundancia es la negación del rendimiento, dándonos como resultado:

### Escenario 1:

Rendimiento = 0.85684

Redundancia = 0.14316

### Escenario 2:

Redundancia = 0.84632

Redundancia = 0.15368

### Escenario 3:

---

Rendimiento = 0.79175

Redundancia = 0.20824

A partir de estos datos se puede deducir que la redundancia en cada uno de los escenarios es bastante baja, por lo tanto el nivel de información que aportan los códigos y su eficiencia son altos. También se puede concluir que el escenario 1 es el caso que aporta mayor información, ya que tiene menor redundancia frente al resto y mayor rendimiento. El rendimiento no es máximo en ninguno de los casos debido a que ninguno de ellos cumple con la condición de que la longitud media sea igual a la entropía de la fuente (aunque están muy cerca).

## Codificación Huffman

Para llevar a cabo la reconstrucción del archivo utilizamos el método de compresión de Huffman.

En la clase Huffman se encuentran los métodos para crear el árbol de Huffman y la tabla que usaremos para la reconstrucción del archivo.

Para la creación del árbol usamos una cola de prioridad, en la que agregamos los nodos del árbol.

```
class Nodo {  
    frecuencia;  
    símbolo;  
    Nodo izq;  
    Nodo der;  
}
```

Luego de tener todos los nodos hoja en la cola, sacamos los dos nodos de menor frecuencia de la cola y creamos un nuevo nodo con dichos nodos como hijos, y con la suma de sus frecuencias. Este nuevo nodo también es agregado a la cola. Lo repetimos hasta que solo quede un nodo en la cola, que será la raíz del árbol de Huffman.

```
mientras (tamaño de la cola > 1) {  
  
    Nodo der := sacaCola  
    Nodo izq := sacaCola  
    temp := nuevo Nodo  
  
    frecuencia := frecuencia derecha + frecuencia izquierda  
    símbolo := "-"  
    izq de temp := izq  
    der de temp := der  
    raíz := temp  
}
```

```
    agregar Cola(temp)
}
```

La tabla que contiene el código de Huffman es un Hashmap, cuya key es el símbolo y el valor el código.

```
public Map <String, String> tablaHuffman = new HashMap<String, String>();
```

Se recorre el árbol, y cada vez que se va a la izquierda se agrega un 0 al código, y cada vez que se va a la derecha se agrega un 1. Cuando el nodo es hoja, se agrega a la tabla el símbolo con su código de Huffman.

```
ALGORITMO carga Tabla( raíz , código);

    if (izquierda de raiz == null and derecha de raiz == null ) {
        agrego a la tabla el símbolo y el código
        return
    }
    cargaTabla(izquierda de raiz, codigo + "0")
    cargaTabla(derecha de raiz, codigo + "1")
```

### Conclusión

La cantidad de caracteres después de reconstruir el archivo con el código de Huffman es mayor al de la fuente original en los 3 casos. Para palabras de :

- **3 caracteres:** cantidad de caracteres del archivo 13684
- **5 caracteres:** cantidad de caracteres del archivo 13474
- **7 caracteres:** cantidad de caracteres del archivo 12623

Por lo que podemos suponer que con palabras de mayor longitud, para esta cantidad de caracteres de la fuente, se podría esperar una compresión mayor. Sin embargo, si tomamos en cuenta que el archivo original es de 10000 bytes, para palabras de :

- **3 caracteres:** archivo binario de 1710.5 bytes
- **5 caracteres:** archivo binario de 1684.25 bytes
- **7 caracteres:** archivo binario de 1577.875 bytes

Lo que significa una compresión aproximadamente del 83% en los 3 casos.

---

## Conclusión final

En este trabajo creamos un programa para realizar los análisis correspondientes. A partir de la fuente provista por la cátedra, calculamos probabilidades condicionales y determinamos que es una fuente de memoria nula, ya que la diferencia de probabilidad condicional es despreciable,. Generamos la extensión de orden 20 y corroboramos la propiedad de entropía para una fuente de extensión N. En la segunda parte analizamos los códigos de longitud 3, 5 y 7 de la fuente propuesta, calculando la cantidad de información que proporciona cada palabra y la cantidad promedio de información de la fuente (entropía) en cada caso, llegamos a la conclusión de que la cantidad promedio de información aumenta mientras la palabra es más grande, pero no de manera proporcional. Siguiendo con los análisis de los tres escenarios, dimos a conocer de qué tipo son, resultando que son no singulares, unívocamente decodificables, y el código de 5 palabras es el único instantáneo. Luego estas conclusiones y análisis se reforzaron con los cálculos de las inecuaciones de Kraft, MacMillan y la longitud media.

Por último, para cada uno de los escenarios mencionados aplicamos el método de codificación de Huffman, y reconstruimos el archivo para cada codificación obtenida.