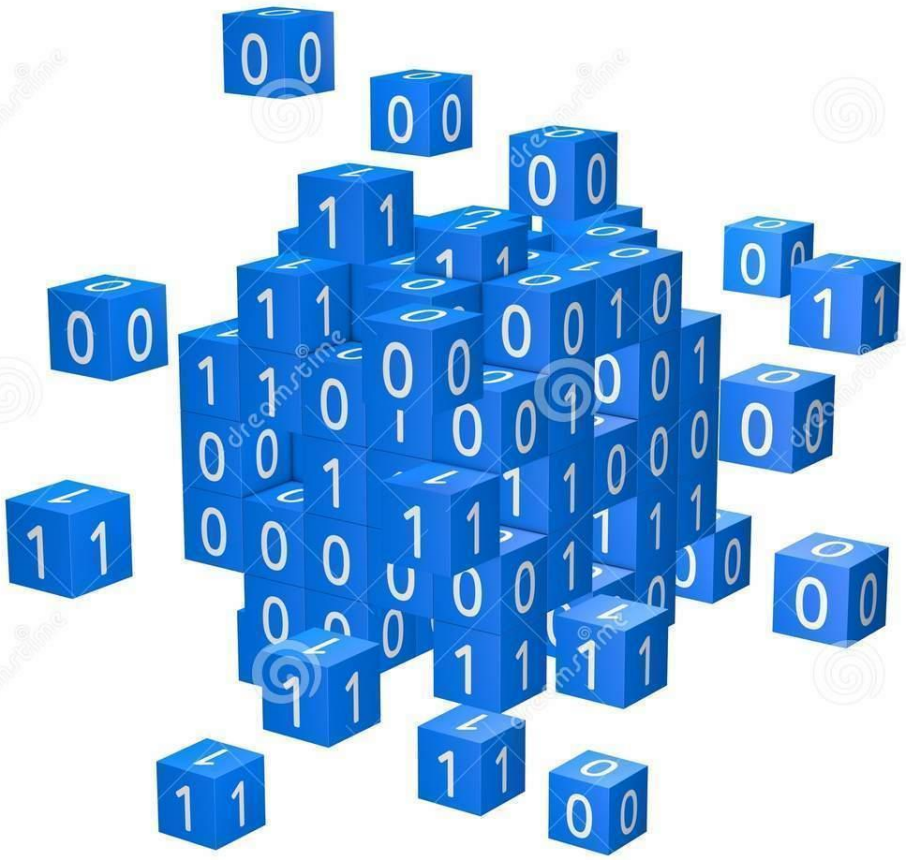
TRABAJO INTEGRADOR Nº1



Integrantes del grupo:

- Alan Gutiérrez, legajo: 13172, e-mail: alan.gutierrez@alumnos.fi.mdp.edu.ar
- Francisco Stimmler, legajo: 15409, e-mail: franciscostimmler@yahoo.com.ar
- María Josefina Oller, legajo: 11609, e-mail: josefinaoller19@gmail.com

Link del repositorio: https://github.com/JosefinaOller/TeoriaDeLaInformacion

Fecha de entrega: 20 de octubre de 2022

Índice

Resumen

En este informe se tratará sobre el análisis y resolución de diversos problemas que surgen al enfrentarse a la codificación y transmisión de la información. Se desarrollarán las técnicas y algoritmos utilizados para su resolución, y se harán conclusiones respecto a los resultados obtenidos. Se abordarán las temáticas de:

- Fuente de información de memoria nula;
- Codificación de símbolos de los códigos y sus propiedades.

Introducción

Antes de abordar los aspectos relevantes de las temáticas mencionadas anteriormente, se explicará la terminología necesaria para comprender el análisis que se realizará durante el informe.

Concepto de Información

Vamos a decir que un evento o un acontecimiento aporta información cuando éste se refiere a un hecho desconocido de antemano. Durante el desarrollo, se va a analizar que, desde el punto de vista probabilístico, cuanto menor sea la probabilidad de ocurrencia de un suceso, mayor será la información que éste brindará.

Concepto de fuente de Información

Llamaremos así a una fuente conformada de símbolos o sucesos, los cuales en su conjunto son denominados *"el alfabeto"* de la fuente, cada uno con una probabilidad de ocurrencia. Al transmitir información, en cada instante de tiempo, la fuente elige según su distribución de probabilidades de un símbolo para ser emitido y transmitir la información que éste posea. La información transmitida durante el desarrollo se contabilizará en la unidad *binit*, la cual viene del uso de *"biestables"* al representar la información. Teniendo en cuenta estos conceptos previamente explicados, se abordarán las diferentes temáticas mencionadas al comienzo del informe.

Desarrollo

Durante esta etapa, se analizará el comportamiento de la fuente proporcionada por la cátedra, obteniendo resultados teóricos y prácticos a analizar. Las tablas de los valores obtenidos se mostrarán en la sección del apéndice del informe. También los resultados teóricos estarán en la sección del apéndice.

Se analizará con más detalle la cantidad de información que aportan los distintos tipos de símbolos, y la entropía, la cual representa la cantidad de información media por símbolo que emite una fuente.

Fuente de memoria nula

En este tipo de fuentes de información, la probabilidad de ocurrencia de cada suceso, o aparición de un símbolo del alfabeto, es independiente de los símbolos previamente obtenidos, es decir, los símbolos son estadísticamente independientes unos a otros.

Se analizó la fuente de información proporcionada por la cátedra, de la cual se extrajo ciertos resultados, y además se compararon los valores teóricos contra los obtenidos, obteniendo así conclusiones, las cuales se muestran a continuación.

Cálculo y análisis: Probabilidades condicionales

La probabilidad condicional se refiere a la probabilidad de que ocurra un evento A, sabiendo que también sucede otro evento B. Entonces para calcular las probabilidades condicionales, primero se genera un arreglo de datos para guardar todos los caracteres leídos del archivo proporcionado por la cátedra para luego generar una matriz de pasaje o de transición de estados. A continuación, se muestra la manera para acumular las apariciones de un símbolo con el otro:

```
datos[]; //arreglo de datos
Para cada símbolo i + 1 hasta la cantidad total de caracteres {
    Si el elemento i de datos[] es igual a 'A' entonces{
        * Si el elemento anterior de datos[] es igual a 'A' entonces
            Aumenta en 1 la frecuencia del elemento [0][0] de la matriz
        Sino
            *Si el elemento anterior de datos[] es igual a 'B' entonces
                Aumenta en 1 la frecuencia del elemento [0][1] de la matriz
            Sino
                *Si el elemento anterior de datos[] es igual a 'C' entonces
                    Aumenta en 1 la frecuencia del elemento [0][2] de la matriz
                Sino    Salta error }
        Sino
          Si el elemento i de datos[] es igual a 'B' entonces{
            //Se repite exactamente los mismos pasos marcados con * , aumentando las
            frecuencias de los elementos [1][0],[1][1],[1][2] de la matriz.  }
          Sino
           Si el elemento i de datos[] es igual a 'C' entonces {
            //Se repite exactamente los mismos pasos marcados con * , aumentando las
        frecuencias de los elementos [2][0],[2][1],[2][2] de la matriz.  }
          Sino Salta error }
```

Una vez generada la matriz de apariciones, se va a realizar la generación de una matriz de pasaje o de transición de datos. Dicha generación se realiza a través del pseudocódigo:

```
Para la posición i hasta la cantidad de elementos de la matriz de apariciones {
    Para la posición j hasta la cantidad de elementos de la matriz de apariciones {
        Elemento [j][i] de la matriz de pasaje = elemento[j][i] de la matriz de
        apariciones/cantidad de veces que aparece el símbolo i en el archivo}}
```

De esa manera, se obtuvieron las probabilidades condicionales del alfabeto mostradas en el apéndice.

Cálculo y análisis: Verificación de memoria nula

Para saber si la fuente es de memoria nula o no, se calculó la matriz de pasaje o de transición de estados(matrizPasaje[][] en el pseudocódigo). De ahí, se obtienen el mayor y el menor de cada columna y se obtiene la diferencia. Lo mismo se hace para las diferencias y si el resultado de la diferencia mayor y la menor es menor a 0.02 se puede afirmar que es de memoria nula. Caso contrario, se afirma que es de memoria no nula.

```
Algoritmo memoriaNula(){
    Constantes: N = 3;
    Variables: Real dMayor = 0.0; Real dMenor = 1.0; Real matrizPasaje[][];
    Para (int j = 0; j < N; j++) hacer
        Real mayor = 0.0; Real menor = 1.0; Real diferencia = 0.0;
        Para (int i = 0; i <N; i++)
            Si (matrizPasaje[j][i]>mayor) entonces
                mayor=matrizPasaje[j][i];
            Fin Si
            Si (matrizPasaje[j][i]<menor) entonces
                menor=matrizPasaje[j][i];
            Fin Si
        Fin Para
        diferencia = mayor-menor;
        Si (diferencia>dMayor) entonces
            dMayor=diferencia;
        Fin Si
        Si (diferencia<dMenor) entonces
            dMenor=diferencia;
        Fin Si
    Fin Para
    Si (dMayor-dMenor<=0.02)
        Escribir: "Es una fuente de memoria nula";
    Sino
        Escribir: "Es una fuente de memoria no nula";
    Fin Si
Fin Algoritmo
```

De esa manera, se realizó la verificación de memoria nula y en nuestro caso, se verificó que es **memoria nula**. Para comprobar si esta verificación es correcta, calculamos teóricamente la suma de cada columna de la matriz de pasaje (adjunta en el apéndice) generada anteriormente.

Resultados teóricos

Suma de la columna 1: **0,99969** Suma de la columna 2: **1** Suma de la columna 3: **1**

Resultados prácticos

Suma de la columna 1: **0,99968** Suma de la columna 2: **0,99999** Suma de la columna 3: **1**

Al comparar los resultados teóricos y prácticos, podemos concluir que la verificación de memoria nula es correcta, ya que cada suma de las columnas es menor o igual a 1, condición necesaria para determinar si una fuente de memoria es nula. Como se comprobó que es una fuente de memoria nula, se va a calcular la entropía de la fuente inicial y la de orden 20. Para eso, vamos a mencionar los siguientes aspectos necesarios para calcular la entropía.

Cálculo y análisis: Cantidad de información

Para calcular la entropía, es necesario calcular la cantidad de información de cada símbolo. Cada símbolo del alfabeto de una fuente tiene cierta posibilidad de transmitir información. La cantidad de información de un símbolo depende exclusivamente de su probabilidad de aparición. A continuación, se muestra el pseudocódigo en donde se calculó la cantidad de información que transmite cada símbolo.

Para cada símbolo i { //Suponiendo que todas las P (ocurrencia del símbolo i)! = 0

Cantidad de información que transmite el símbolo i = -log2$(P(ocurrencia))$ *}*

Mediante este pseudocódigo, obtuvimos la cantidad de información que transmite cada símbolo, y para comprobar que estos valores obtenidos son correctos, vamos a calcular nuevamente la cantidad de información de manera teórica utilizando las probabilidades teóricas mostradas en el apéndice:

I experimental (A) = 1,646 binits	I teórica (A) = 1,644 binits
I experimental (B) = 1,585 binits	I teórica (B) = 1,599 binits
I experimental (C) = 1,525 binits	I teórica (C) = 1,514 binits

Por el análisis de los valores obtenidos anteriormente, podemos determinar que la información transmitida por los símbolos es correcta y, como aspecto importante a mencionar, cuanto menor sea la probabilidad de ocurrencia de un símbolo, mayor será la información que éste transmite. Este aspecto se puede evidenciar al hacer la comparación de los valores obtenidos para los símbolos "A", "B" y "C" respectivamente. Sin embargo, es importante aclarar que, en el caso que un símbolo tenga posibilidad de ocurrencia cero, la cantidad de información que transmite también será cero, ya que, se sabe anteriormente que nunca va a ser transmitido.

Cálculo y análisis: Entropía

La entropía de una fuente depende directamente de la cantidad de información que transmiten los símbolos pertenecientes a ella. Por lo tanto, la forma de calcular la entropía de una fuente es la siguiente:

entropía = 0

Para cada símbolo i { //Suponiendo que todas las P (ocurrencia del símbolo i)! = 0

*entropía = entropía + cantidad de información que transmite el símbolo i * probabilidad del símbolo i }*

Después de realizar dicho cálculo, se obtuvo la entropía de la fuente inicial:

H(S)=1,584 [binits/símbolo]

Según el apéndice, el valor teórico de la entropía de la fuente inicial es:

H(S)=1,584 [binits/símbolo]

Para la extensión de orden 20, se pensó en 20 ciclos *for* anidados y se llegó al valor de 31,682281 [binits/símbolo]. El valor teórico es 31,682281 [binits/símbolo] y se llega al multiplicar 20×la entropía inicial del alfabeto.

$H(S^n) = nH(S)$, con n=20.

Redondeando a 6 decimales se llegó al mismo resultado de la entropía de orden 20. Por lo tanto, podemos concluir que los valores de la entropía son correctos.

A partir de ahora, tomando el archivo precedente, vamos a considerar que las cadenas representan palabras de un código, de 3, 5 y 7 caracteres.

Cálculo y análisis: Generación de palabras de N caracteres en un código

Se agarra el archivo y por cada cantidad N de caracteres se agrega a la lista de palabras.

```
procesamiento(cantCaracteres ){
        i = 0
        mientras(i sea menor que cantCaracteresArchivo){
                j=i
                mientras( j sea menor que  (i+cantCaracteres){
                        si( j es menor que  cantCaracteresArchivo )
                                palabra=palabra+datos[j]
                        j=j +1}
                si( largo de palabra es igual que cantCaracteres)
                        se añade palabra al arreglo de palabras
        i=i+cantCaracteres }
```

De esta manera, obtenemos un listado con todas las palabras de N caracteres dentro del archivo, cabe destacar que si la cantidad de caracteres del archivo no es divisible por N los caracteres de sobra son ignorados.

Cálculo y análisis: Frecuencia de palabras de N caracteres en un código

Primero se genera un arreglo de datos para guardar todos los caracteres leídos del archivo proporcionado por la cátedra y se agrupan esos caracteres de a N cantidad en palabras y se calcula las frecuencias de la siguiente manera:

Palabras[]; //arreglo de palabras

Apariciones[]; //HashMap de frecuencias (clave,frecuencias)

Por cada palabra de N longitud i hasta la cantidad total de caracteres {

Si (la clave de apariciones contiene al palabras[i])

Se acumula +1 al valor de frecuencias de Apariciones[]

Sino

Se pone 1 al valor de frecuencias de Apariciones[]}

De esa manera, se obtuvieron las frecuencias de las palabras de N caracteres, las cuales se muestran en el apéndice junto con sus palabras.

Cálculo y análisis: Cantidad de información en palabras de N caracteres en un código

En este caso, el pseudocódigo para calcular la cantidad de información que transmite una palabra de N caracteres es igual al de cálculo y análisis de cantidad de información explicado anteriormente, la única diferencia es que, en vez de aplicar la fórmula $-\log_2(P(ocurrencia))$, se debe utilizar la siguiente fórmula $-\log_N(P(ocurrencia))$. Los resultados de la cantidad de información que transmite cada palabra de N caracteres se muestran en el apéndice. Como se explicó anteriormente, nuevamente podemos decir que los resultados son correctos.

Cálculo y análisis: Entropía en palabras de N caracteres en un código

Al igual que el caso anterior de la cantidad de información, el pseudocódigo para calcular la entropía es igual al de cálculo y análisis de la entropía explicado anteriormente. Mediante dicho pseudocódigo, se obtuvieron los resultados de la entropía total de palabras de N caracteres:

H (S de 3 caracteres) =**2,971159** [binits/símbolo]

H (S de 5 caracteres) =**4,35374** [binits/símbolo]

H (S de 7 caracteres) =**4,48432** [binits/símbolo]

Como ya se analizó la comprobación de datos correctos anteriormente, nuevamente podemos decir que los resultados son correctos.

Dado que en todos los casos las palabras son todas de un mismo largo N, ninguna palabra del código coincide con el prefijo de otra, eso en condición necesaria y suficiente para que un código sea instantáneo, por lo tanto, en todos los casos obtenemos un **código instantáneo.**

Cálculo y análisis: Inecuación de Kraft, McMillan, Longitud Media

Los siguientes pseudocódigos de la inecuación de Kraft y la longitud media se muestran a continuación:

desigualdad_de_kraft = 0

Para cada palabra i {

desigualdad_de_kraft = desigualdad_de_kraft + (catidad_de_simbolos_diferentes elevado a -(largo de la palabra i)}

SI desigualdad_de_kraft <= 1 Entonces: Escribe "Cumple la desigualdad de kraft, por lo tanto es instantáneo."

Sí no Escribe "No cumple la desigualdad de kraft, no es instantáneo."

longitud_media = 0

Para cada palabra i {

*longitud_media = longitud_media + Probabilidades[i] * largo de la palabra i}*

En el caso de un S de 3, 5 y 7 la desigualdad de kraft da 1 es natural que de ese valor al ser todas la palabras de la misma longitud,Sólo es una medida cuantitativa ya que dice que es posible pero no asegura que el código que armamos sea instantáneo, pero al no existir un prefijo común y ser códigos unívocos podemos nuevamente confirmar que se tratan de **códigos instantáneos.**

Obviamente la longitud media nos da igual que la longitud de las palabras, en el caso de 3 y 5 caracteres nos da una entropía cercana a la cantidad de caracteres, todos los códigos unívocos que pueden aplicarse a la misma fuente y el mismo alfabeto no podrán ser menores que la longitudes que tienen, por lo cual al no poder tener menor la cantidad de caracteres por palabra podemos decir que los códigos son **compactos.**

En cambio, en el caso de los 7 caracteres obtenemos una entropía de 4,48432 [binits/símbolo] por lo cual puede aplicarse un código unívoco a la misma fuente con una longitud de 5 caracteres, por eso podemos decir que se trata de un **código no compacto.**

Cálculo y análisis: Redundancia y Rendimiento

Definimos el rendimiento como la división entre H(S) y la longitud. Y a redundancia como el opuesto 1-rendimiento.

rendimiento = entropia/longitudMedia

redundancia = 1 - rendimiento

En el caso de S de 3 caracteres → Rendimiento = 0,9904 Redundancia = 0,0096

En el caso de S de 5 caracteres → Rendimiento = 0,8707 Redundancia = 0,1293

Para el caso de 3 caracteres y el caso de 5 caracteres, obtenemos un buen rendimiento de valores casi máximos, mientras que una redundancia da valores casi nulos.

En el caso de S de 7 caracteres → Rendimiento = 0,6404 Redundancia = 0,3594

En este caso, obtenemos un rendimiento mayormente alto, pero no del todo satisfactorio junto con una redundancia baja pero no tanto.

Podemos concluir que en 3 y 5 caracteres tenemos un buen rendimiento y baja redundancia mientras que en el de 7 se podría mejorar el rendimiento y bajar la redundancia.

Cálculo y análisis: Codificación de Huffman

Para la codificación se decidió usar el método Huffman porque se necesitaba obtener códigos binarios para después comprimir un archivo. Para obtener los códigos de Huffman se construyó un árbol binario de nodos, a partir de una lista de nodos, cuyo

tamaño depende del número de símbolos. Los nodos contienen dos campos, las claves, ya sean de 3, 5 o 7 caracteres, y la probabilidad. El algoritmo consiste en la creación de un árbol binario que tiene cada uno de los símbolos por hoja, y construido de tal forma que siguiéndolo desde la raíz a cada una de sus hojas se obtiene el código Huffman asociado a él.

1. Se crea una lista de árboles, uno por cada uno de los símbolos del alfabeto, consistiendo cada uno de los árboles en un nodo sin hijos, y etiquetado cada uno con su símbolo asociado y su frecuencia de aparición.
2. Se toman 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. Esos dos árboles se eliminan de la lista. También se etiquetan las dos ramas del nuevo árbol: con un 0 la de la izquierda, y con un 1 la de la derecha.
3. Se repite el paso 2 hasta que solo quede un árbol.

El siguiente pseudocódigo donde se realiza el algoritmo Huffman:

```
Algoritmo Huffman
        Lista lista;//lista de árboles Lista list;//lista de probabilidades
        Entero cantCaracteresCodigo;//depende si es de 3, 5 o 7 caracteres
Llamar NuevaLista(lista)
        //Genero una lista con las probabilidades y la ordeno
        Llamar NuevaLista(list)
        Llamar OrdenarLista(list)
        Para cada elemento de list hacer
                Nodo nodo = Llamar NuevoNodo(list.clave, list.valor, nulo, nulo);
                Llamar AgregarNodo(Lista, nodo);
        Fin Para
        Mientras lista.tamano!=1 hacer
                Entero i = 0; Entero j = 1; Real min1=1; Real min2=1;
                Para k = 0 hasta lista.tamano hacer
                        Si lista.k.Probabilidad<min1 entonces
                                min2 = min1;
                                j = i;
                                min1 = lista.k.Probabilidad;
                                i = k;
                        Sino
                                Si lista.k.Probabilidad<min2 entonces
```

```
                                        min2 = lista.k.Probabilidad;
                                        j = k;
                                    Fin Si
                                Fin Si
                            Fin Para
                            nodo = Llamar NuevoNodo(lista.i.Clave+lista.j.Clave, lista.i.Probabilidad
+ lista.j.Probabilidad, lista.i, lista.j)
Llamar AgregarNodo(Lista, nodo);
                            Si i < j entonces
                                Llamar EliminarNodo(Lista, j);
Llamar EliminarNodo(Lista, i);
                            Sino
                                Llamar EliminarNodo(Lista, i);
Llamar EliminarNodo(Lista, j);
                            Fin Si
                        Fin Mientras
                        Llamar recorrido(lista.0);
                        Llamar codificacion(cantCaracteresCodigo));
Fin Algoritmo
```

Al codificar los símbolos se puede afirmar que es posible comprimir un archivo de 10000 bytes en uno que ocupa menos de 100, ya que al aplicar el algoritmo de Huffman, los símbolos más frecuentes tienen una longitud de código más corta que uno menos frecuente.

Conclusiones

En este trabajo práctico, se realizaron cálculos de probabilidades condicionales, entropía de la fuente inicial, entropía de orden 20 y la verificación de memoria nula, y también se calcularon dichos cálculos para palabras código de 3,5 y 7 caracteres. Además, se hicieron los cálculos de la Inecuación de Kraft, McMillan, Longitud Media, la verificación de códigos compactos y por último, la codificación de Huffman. Para ello, se utilizaron los conceptos teóricos y prácticos vistos en la asignatura. Las conclusiones ya fueron desarrolladas en la sección anterior durante el desarrollo del informe.

Apéndice

En el siguiente apéndice se incluyen todas los resultados, así como todos los resultados teóricos realizados durante el desarrollo del informe.