

UNIVERSIDAD DE LA REPÚBLICA

FACULTAD DE INGENIERÍA

COMUNICACIONES DIGITALES

---

## Taller 3 | Teoría de la Información: Codificación

---

*Autores:*

Federico BELLO

Julieta UMPIERREZ

28 de septiembre de 2025



UNIVERSIDAD  
DE LA REPÚBLICA  
URUGUAY



# Índice

<b>1. Parte I: Canal simétrico con borrado</b>	<b>2</b>
1.1. Parte a . . . . .	2
1.2. Parte b . . . . .	2
1.3. Parte c . . . . .	2
1.4. Parte d . . . . .	2
1.5. Parte e . . . . .	3
1.6. Parte f . . . . .	3
1.7. Parte g . . . . .	4
<b>2. Parte II: Algoritmos de decisión iterativa</b>	<b>4</b>
2.1. Parte a . . . . .	4
2.2. Parte b . . . . .	4
2.3. Parte c . . . . .	5
2.4. Parte d . . . . .	5
2.5. Parte e . . . . .	5
<b>3. Parte III: Compresión de la fuente</b>	<b>6</b>
3.1. Parte a . . . . .	6
3.2. Parte b . . . . .	6
3.3. Parte c . . . . .	8
3.4. Parte d . . . . .	8
<b>4. Parte IV: Codificación para corrección de errores</b>	<b>9</b>
4.1. Ejercicio 9.4 . . . . .	9
4.1.1. Parte a . . . . .	9
4.1.2. Parte b . . . . .	9
4.2. Ejercicio 9.6 . . . . .	11
4.3. Ejercicio 9.7 . . . . .	12

## 1. Parte I: Canal simétrico con borrado

### 1.1. Parte a

Siguiendo las probabilidades condicionales de la letra se tiene que utilizando probabilidad total  $p(e) = P(X = x_0, Y = e) + P(X = x_1, Y = e) = P(X = x_0)P(Y = e|X = x_0) + P(X = x_1)P(Y = e|X = x_1) = ap + (1 - a)p$ . En resumen  $p(e) = ap + (1 - a)p = p$

### 1.2. Parte b

Los bits de información por símbolo transmitido se encapsulan en  $\mathbb{H}\{X\} = -[P(X = x_0) \log(P(X = x_0)) + P(X = x_1) \log(P(X = x_1))] = -[a \log(a) + (1 - a)\log(1 - a)] = \Omega(a)$ . En la figura 1 se adjunta la gráfica de la misma en la que se puede ver como tiene forma de parábola.

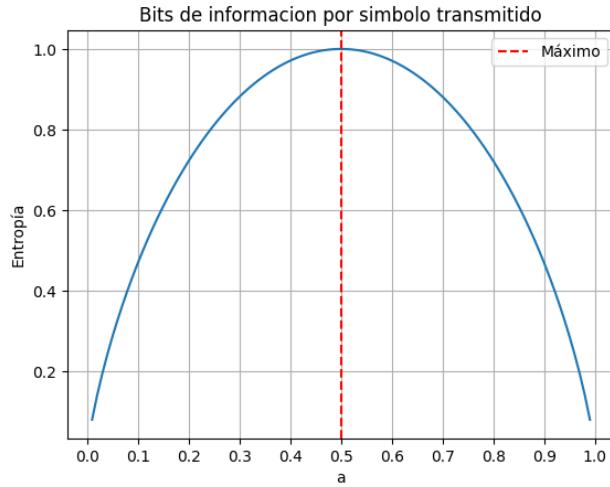


Figura 1: Entropía de  $X$

### 1.3. Parte c

Derivando e igualando a 0 la expresión anterior es trivial ver que el máximo se da en  $a = 1/2$  lo que es a su vez muy intuitivo ya que implica que la mayor incertidumbre esta al tener una situación equiprobable. Este valor se marca en rojo en la gráfica de la figura 1.

### 1.4. Parte d

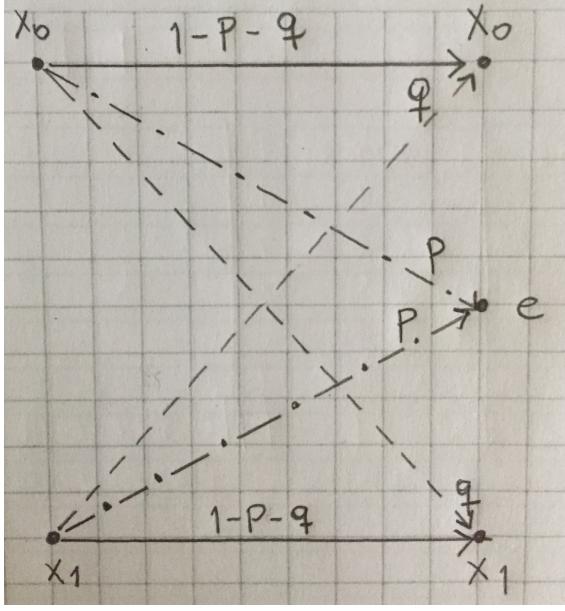
Para modificar el sistema BPSK en recepción se debería agregar una zona intermedia (de tamaño a diseñar) en la cual el receptor responde 'no se' y se borra lo recibido. El diagrama de BPSK adaptado a esta situación se adjunta en la figura 2a.

Para calcular  $p$  y  $q$  se utilizo el diagrama de la figura 2b tomando  $x_0$  en  $-A$  y  $x_1$  en  $A$ . Además se considera ruido  $\mathcal{N}(0, \sigma^2)$

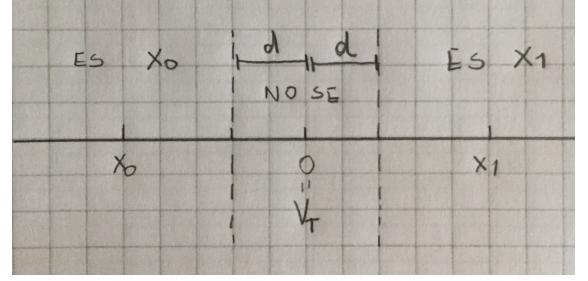
$$q = P(Y = x_0 | X = x_1) = P(Y = x_1 | X = x_0) \stackrel{(1)}{\approx} P(A + \text{ruido} < -d) = P(\text{ruido} < -d - A) = Q\left(\frac{A + d}{\sigma}\right)$$

donde (1) es tomando el caso  $P(Y = x_0 | X = x_1)$ .

Análogamente se calcula  $p$



(a) BPSK modificado para tener borrado



(b) Zonas de decisión para BPSK con borrado

**Figura 2:** BPSK con borrado

$$p = P(Y = e|X = x_1) = P(Y = e|X = x_0) = P(-A + \text{ruido} \in [-d, d]) \stackrel{(2)}{\approx} P(A - d < \text{ruido} < A + d)$$

$$p = Q\left(\frac{A-d}{\sigma}\right) - Q\left(\frac{A+d}{\sigma}\right) \quad (1)$$

donde (2) es usando  $P(Y = e|X = x_0)$ .

### 1.5. Parte e

Si, dado que si no lo asumimos el sistema funcionaría muy mal dado que  $q$  es la probabilidad de pasarse de la zona de borrado, lo que implica estar muy lejos del símbolo correcto.

### 1.6. Parte f

Para hacer esos cálculos es necesario conocer las siguientes probabilidades  $P(Y = x_0) = a(1-p)$ ,  $P(Y = e) = p$  y  $P(Y = x_1) = (1-a)(1-p)$ .

Utilizando la definición y que  $q = 0$  se tiene entonces que

$$\mathbb{H}\{Y\} = \mathbb{E}[-\log P(y)] = -[a(1-p)\log(1-a)(1-p) + (1-a)(1-p)\log(1-a)(1-p) + p\log p] = \Omega(p) + (1-p)\Omega(a)$$

$$\mathbb{H}\{Y|X\} = \mathbb{E}[-\log P(y|x)] = -[(1-p)\log(1-p) + p\log p] = \Omega(p)$$

Recordando que  $I\{X; Y\} = \mathbb{H}\{Y\} - \mathbb{H}\{Y|X\} = (1-p)\Omega(a)$

Y recordando que  $\mathbb{H}\{X|Y\} = \mathbb{H}\{X\} - I\{X; Y\} = p\Omega(a)$

$\mathbb{H}\{Y\}$  me da los bits de información por símbolo recibido, luego  $\mathbb{H}\{Y|X\}$  es la información que me da  $Y$  sabiendo el  $X$  que se envió,  $\mathbb{H}\{X|Y\}$  es la incertidumbre de  $X$  dado que vimos  $Y$  y  $I\{X; Y\}$  es la reducción de incertidumbre de una variable aleatoria al conocer otra.

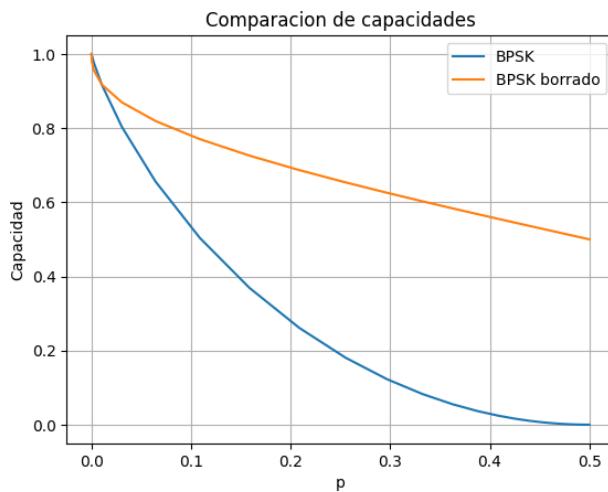
Para calcular la capacidad del canal debemos maximizar la información mutua eligiendo  $a$  que es la probabilidad con la que mando cada símbolo.

$$\max_a I\{X; Y\} = \max_a (1-p)\Omega(a)$$

La solución se da en  $a = \frac{1}{2}$  y el valor funcional es  $1 - p$ .

### 1.7. Parte g

La capacidad del canal BPSK es  $C_{BPSK} = 1 - \Omega(p_{BPSK})$  con  $p_{BPSK} = Q\left(\frac{A}{\sigma}\right)$  y la de BPSK con borrado es  $C_{BPSK_{Borr}} = 1 - p$  con el  $p$  hallado en la parte d. Luego se definió que  $d = 0,216A$  por lo que las probabilidades de error en ambos casos quedaron expresados en los mismos términos. Suplantando  $d$  en la ecuación 1 para dejar ambas expresiones en función de  $\frac{A}{\sigma}$  se llega a la gráfica mostrada en la figura 3. En la misma se ve como introducir el borrado aumenta la capacidad, dado que no se tiene incertidumbre cuando llega un símbolo valido, ya que si el símbolo llega en la zona de borrado se descarta y se asume que no van a llegar por fuera de la misma (y del otro lado).



**Figura 3:** Comparación de capacidades entre BPSK y BPSK con borrado

## 2. Parte II: Algoritmos de decisión iterativa

### 2.1. Parte a

$\mathbb{H}\{X|Y_k\}$  representa la incertidumbre de  $X$  luego de haber visto el patrón  $Y_k$ . Visto de otra manera, se puede pensar  $\mathbb{H}\{X\} - \mathbb{H}\{X|Y_k\}$  como la información sobre  $X$  que brinda haber visto  $Y_k$ .

### 2.2. Parte b

Parece claro que el patrón resultante depende solamente de la palabra oculta y de la palabra ingresada. Es decir, dada que uno sabe la palabra oculta y la palabra ingresada, es posible calcular el patrón. Por lo tanto,  $Y_k = f(X, \mathbf{x}_k)$ .

### 2.3. Parte c

Planteando la entropía conjunta, se obtiene que  $\mathbb{H}\{X; Y_k\} = \mathbb{H}\{Y_k|X\} + \mathbb{H}\{X\} = \mathbb{H}\{X|Y_k\} + \mathbb{H}\{Y_k\}$ , por lo tanto,  $\mathbb{H}\{X|Y_k\} = \mathbb{H}\{Y_k|X\} + \mathbb{H}\{X\} - \mathbb{H}\{Y_k\}$ . Teniendo en cuenta las dos partes anteriores parece claro que el termino  $\mathbb{H}\{X|Y_k\}$  es el que desea minimizar, ya que, dada una entrada la cual elige el jugador, se desea maximizar la información que se conoce sobre la palabra oculta, o, lo que es lo mismo, minimizar la incertidumbre.

Sobre el resultado anterior, observar dos cosas importantes. La primera es que el jugador no tiene control sobre el termino  $\mathbb{H}\{X\}$ , esta dado por la distribución de probabilidad de las palabras ocultas, por lo que no es posible modificarlo. Además, es fácil ver que el termino  $\mathbb{H}\{Y_k|X\}$  es cero. Esto, se debe a que  $Y_k = f(X, \mathbf{x}_k)$ , donde  $\mathbf{x}_k$  es una variable determinística, no tiene incertidumbre. Por lo tanto, la incertidumbre del patrón dado que conocemos la palabra oculta y la determinística es cero, el patrón que genera ya está dado. Para verlo matemáticamente alcanza con plantear la definición de la entropía condicional:  $\mathbb{H}\{Y_k|X\} = \mathbb{E}\{-\log(\{Y_k|X\})\} = \sum_i \sum_j -P(Y_k = y_j|X = x_i) \log P(Y_k = y_j|X = x_i)$ , donde  $P(Y_k = y_j|X = x_i)$  es 0 o 1  $\forall i, j$ , ambos valores donde la entropía es nula.

Retomando el resultado original, se llego a que  $\mathbb{H}\{X|Y_k\} = \mathbb{H}\{X\} - \mathbb{H}\{Y_k\}$ , siendo claro entonces que para minimizar  $\mathbb{H}\{X|Y_k\}$  es necesario maximizar  $\mathbb{H}\{Y_k\}$ .

### 2.4. Parte d

Para calcular la probabilidad de cada patrón para cierto  $x_k \in \mathcal{X}_k$  primero se supondrá que todas las palabras pertenecientes a  $\mathcal{X}_k$  son equiprobables. La probabilidad entonces de obtener el patrón todo verde es siempre  $\frac{1}{\#\{\mathcal{X}_k\}}$  dado que hay solamente una palabra en el conjunto que genera dicho resultado, por lo que extendiendo el razonamiento, la probabilidad de generar un patrón cualquiera es la cantidad de palabras del conjunto que generan dicho patrón sobre la cantidad de palabras totales de  $\mathcal{X}_k$ . Entonces, si las palabras no fueran equiprobables, es trivial ver que la probabilidad de generar un patrón determinado es simplemente sumar las probabilidades de las palabras que generan dicho patrón.

Luego, una vez que se tiene la distribución de probabilidades para calcular la entropía alcanza con aplicar la definición de la misma.

### 2.5. Parte e

Se corrió el programa brindado con distintas palabras iniciales, corriendo tanto la versión que maximiza la entropía como la que no. Los resultados de las pruebas pueden observarse en la tabla 1, donde se observa claramente como maximizar la entropía resulta en una mejora en el desempeño, disminuyendo la cantidad de intentos promedio independientemente de la palabra inicial. Además, se observa algo intuitivo: cuanto menos información brinde la palabra inicial (palabras con letras muy poco probables) mayor sera la cantidad de intentos. No solo esto, sino que utilizar una palabra que brinda muy poca información es *casi* lo mismo que perder un intento. Notar como la diferencia entre comenzar con *salet* o con *xylyl*<sup>1</sup> es cercana a uno, independientemente de la estrategia utilizada, indicando que la información de esta segunda palabra es *casi* nula.

---

<sup>1</sup>Algo químico

Palabra inicial	Versión	Al azar	Maximizando entropía
Salet		4.65	3.91
Hello		5.03	4.19
Xylyl		5.42	4.55

**Cuadro 1:** Desempeño de ambas versiones con distintas palabras iniciales

Además, el puntaje promedio humano (4.2<sup>2</sup>) parece ser indicador que al jugar se opta por una estrategia mixta entre ambas versiones del algoritmo, lo cual parece cerrar intuitivamente. Por ejemplo, si se inicia con la palabra *slane* y el patrón correspondiente es [*gris, amarillo, gris, gris, gris*] parece absurdo pensar que el próximo intento de alguien sea *xyllyl*, indicando que la elección no es del todo azar, por mas que tampoco sea óptima.

### 3. Parte III: Compresión de la fuente

#### 3.1. Parte a

Para aplicar el algoritmo de Huffman es suficiente y necesario conocer la distribución de probabilidad de los símbolos. Esto trae implícito que los símbolos deben ser independientes entre si, ya que asume un estado 'estático', donde las probabilidades de los distintos símbolos se mantienen constantes, independientemente de lo que haya pasado anteriormente. En otras palabras, Huffman consigue un código óptimo si se trata de una fuente DMS. (*discrete memoryless source*). Parece claro ver que estos no son el caso. Por ejemplo, en el lenguaje natural hay una fuerte dependencia: la letra actual tiene una alta correlación con la letra anterior, si una letra dada fue la 'y' es extremadamente poco probable que la siga otra 'y', mientras que además en la amplia mayoría de los casos es posible predecir la ultima letra de una palabra dada las anteriores. Eventualmente un algoritmo podría hacer uso de estas características, aprovechando así al máximo las propiedades probabilísticas de cada secuencia.

#### 3.2. Parte b

En este caso, la entropía para cada secuencia se calcula tomando las probabilidades empíricas, es decir, la frecuencia con la que aparece cada símbolo en cada archivo. Por lo tanto, la entropía sera máxima si en el archivo hay la misma cantidad de ceros y de unos (o de símbolos distintos en caso de tomar extensiones), mientras que se irá minimizando a medida que comience a desbalancearse la frecuencia de cada símbolo. En el cuadro 2 se observa la entropía de cada archivo para distintas extensiones de la fuente. Notar que cuando la entropía es cercana a uno no es posible comprimir el archivo, dado que implica que cada bit lleva un bit de información, la máxima información posible para un bit. Esto ultimo puede verse en el cuadro 3.

Al tomar distintas extensiones de la fuente, se logra buscar patrones a mayor escala dentro de cada archivo, no viendo solamente las distribuciones a nivel de bit, sino a nivel de bloques. La motivación de esto es que la información guardada no suele ser aleatoria, sino que sigue ciertos patrones. Por ejemplo, al hacer la traducción a bits de un texto, a nivel de bit nunca se encontraran patrones, sin embargo, tomar a nivel de byte es equivalente a ver cada letra como un símbolo distinto.

---

<sup>2</sup>Según <https://wordlestat.com/>

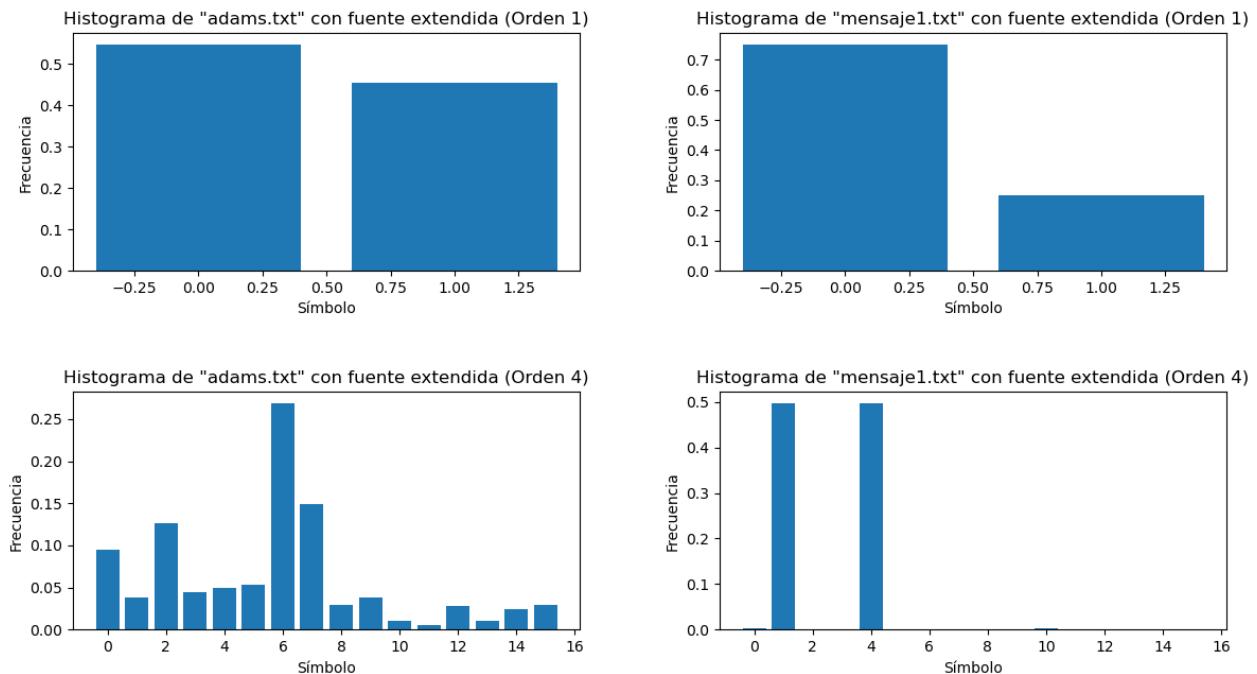
Archivo	Extensión	Sin extender	Orden 2	Orden 4	Orden 8
<i>adams</i>		0.994	0.978	0.843	0.554
<i>mensaje1</i>		0.811	0.513	0.263	0.007
<i>mensaje2</i>		0.999	0.997	0.973	0.700

**Cuadro 2:** Entropía del mensaje para distintas extensiones de la fuente

Archivo	Extensión	Sin extender	Orden 2	Orden 4	Orden 8
<i>adams</i>		0.0	0.0	0.152	0.441
<i>mensaje1</i>		0.0	0.248	0.622	0.874
<i>mensaje2</i>		0.0	0.0	0.018	0.290

**Cuadro 3:** Tasa de compresión para distintas extensiones de la fuente

En la figura 4 se observa la distribución de probabilidad empírica para los archivos 'adams.txt' y 'mensaje1.txt' tomando extensiones de la fuente de orden 1 y 4 a modo de ejemplificar. Comparando con la tabla 2 se observa lo previamente mencionado: cuanto mas uniforme sea la distribución de los símbolos mayor sera la entropía. Comparando con la tabla 3 se observa también como el hecho de que la entropía sea menor implica que es posible comprimir mas, aumentando la entropía del archivo codificado.



**Figura 4:** Distribución de símbolos en distintos archivos

Por ultimo, es importante ver como al intentar comprimir el archivo lo que se logra es un aumento en la entropía del mismo (cuadro 4). Intuitivamente esto se puede ver como que comprimir no es mas que maximizar la información de cada símbolo, buscando eliminar la redundancia, aumentando así la entropía.

Otra forma de ver esto es que: uno puede seguir comprimiendo mientras la entropía no sea máxima, indicando que el archivo 'mensaje1.txt' puede seguir siendo comprimido sin mucha dificultad (entropía 0.052). No es casualidad que sea el que mas lejos esta de cumplir las hipótesis del algoritmo de Huffman.

Archivo \ Extensión	Sin extender	Orden 2	Orden 4	Orden 8
Archivo				
<i>adams</i>	0.994	0.979	0.995	0.995
<i>mensaje1</i>	0.811	0.920	0.920	0.052
<i>mensaje2</i>	0.999	0.997	0.994	0.997

**Cuadro 4:** Entropía del mensaje codificado para distintas extensiones de la fuente

### 3.3. Parte c

Para ilustrar las ideas se utilizara a modo de ejemplo el 'mensaje1.txt', el razonamiento y las conclusiones son idénticas para cualquiera de los archivos. El mismo, es una secuencia de 'A's seguidas de un salto de linea final, por lo que el pasaje a binario queda según la tabla ASCII como: 01000001...01000001 00001010 donde agrupando de a bytes, los primeros corresponden a la 'A' y el ultimo al salto de linea. Notar entonces que dada esta secuencia, si se quisiera hacer una compresión de orden 1, no podría ser comprimido (y efectivamente no lo es), dado que solamente se podría hacer un mapeo de 0,1 en 0,1. En cambio, al tomar una extensión de orden 8 se consigue cambiar el byte entero que representa la 'A' por un bit, y el byte que representa el salto de linea por dos (dado que tiene menor probabilidad). Esto, consigue un tamaño final de archivo casi  $\frac{7}{8}$  menor al original, se utiliza solo un bit para lo que antes se utilizaban 8 (a menos del ultimo byte que se utilizan 2).

Por último, analizando los distintos códigos se observa algo que, aunque esperable, es interesante. Luego de aplicar el algoritmo de Huffman, los archivos que mejor logran comprimirse son aquellos que al tomar la extensión de fuente tienen pocos posibles símbolos. Por ejemplo, si existiese un archivo que tiene en proporciones iguales todas las secuencias de largo  $2^8$ , al tomar la extensión de fuente de largo 8 no sería posible comprimirlo, ya que la entropía del mismo seguiría siendo máxima. Es deseado que exista cierta desproporción al tomar la extensión, y cuanto mayor sea la misma, mayor sera la compresión.

### 3.4. Parte d

En la tabla 5 se observan los largos en bytes de cada archivo, sin comprimir, comprimido con Huffman de orden 8 o utilizando *zip*. Para calcular el largo con Huffman alcanza con utilizar la tasa de compresión de la tabla 3. Notar entonces como aplicar Huffman consigue un resultado al menos comparable con el resultado de *zip*, incluso obteniendo un resultado considerablemente mejor.

Archivo	Compresión	Sin Comprimir	Huffman	<i>zip</i>
<i>adams</i>		1900	1062	1200
<i>mensaje1</i>		167	21	209
<i>mensaje2</i>		55	39	262

**Cuadro 5:** Largo de cada archivo en bytes

Sin embargo, esta comparación no es justa. Al codificar utilizando herramientas de la índole de *zip* no solo se guarda el archivo codificado como tal, sino que también metadata sobre como fue codificado el archivo. A esta metadata se le llama *diccionario* y es útil para luego poder descomprimir. Notar que si se le enviara a alguien el archivo codificado Huffman, no sería capaz de descomprimirlo, necesitando de mas información, en este caso en particular, el código que fue creado al momento de aplicar el algoritmo.

## 4. Parte IV: Codificación para corrección de errores

### 4.1. Ejercicio 9.4

#### 4.1.1. Parte a

Lo que hay que hacer es  $s = yH^T = eH^T$ . Esto da como resultado  $[1, 1, 0]$ . Dado que se sabe que el resultado tiene solo un error, se puede identificar que se da en la coordenada del vector error dado que el resultado de la multiplicación es la segunda fila de  $H^T$ . Si no fuese un solo error podrían ser muchas combinaciones lineales de las filas y así ser mas posiciones en 1 en el vector  $e$ .

#### 4.1.2. Parte b

Para calcular todas las palabras de código validas hay que utilizar  $c = xG$  con  $x$  todos los vectores de tamaño 4 posibles. Para calcular la distancia mínima hay que contar la cantidad de 1s dado que la distancia mínima es la distancia entre dos palabras del código y al pertenecer el nulo al código la distancia mínima va a ser a ese. Ambas cosas se presentan en la tabla 6.

La distancia mínima es 3. Para encontrar la matriz  $H$  hay que encontrar el kernel de  $G$ . Eso implica igualar las filas de  $G$  a 0 y resolver el sistema de ecuaciones.

$$x_1 + x_3 + x_4 = 0$$

$$x_0 + x_2 + x_4 = 0$$

$$x_1 + x_2 + x_5 = 9$$

$$x_0 + x_1 + x_6 = 0$$

El resultado es la siguiente matriz

$$H = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 \end{pmatrix} \quad (2)$$

x	c	distancia
0000	0000000	-
0001	1100001	3
0010	0110010	3
0011	1010011	4
0100	1010100	3
0101	0110101	4
0110	1100110	4
0111	0000111	3
1000	0101100	3
1001	1001101	4
1010	0011110	4
1011	1111111	7
1100	1111000	5
1101	0011001	3
1110	1001010	3
1111	0101011	4

Cuadro 6: Palabras del código y distancia

El síndrome se calcula  $s = rH^T = [1, 0, 0]$ , asumiendo que hay un solo bit errado podemos decir que se dio en la primer coordenada por lo que la palabra enviada era  $[0, 1, 0, 1, 0, 1, 1]$ . Si no asumimos 1 solo error hay muchas posibilidades de palabra enviada.

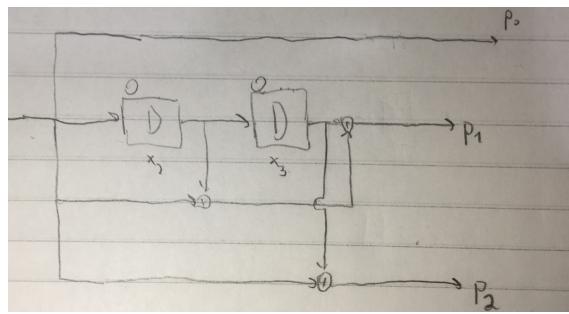
Para llevarla a la forma sistemática la idea es generar a la izquierda una identidad de tamaño  $4 \times 4$ , lo que quede de tamaño  $4 \times 3$  a la izquierda sera  $P$ . Para llevárselo se hacen varios cambios de columnas y luego se suman dos filas para obtener

$$G_{system} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix} \quad (3)$$

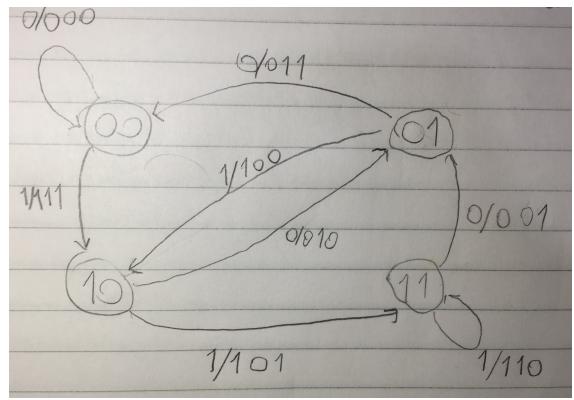
por lo que

$$H_{system} = \begin{pmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{pmatrix} \quad (4)$$

Para escribirlo como filtro la idea es ver  $P$  y viendo un vector de entrada  $[u_1, u_2, u_3, u_4]$  ver los bits de paridad que saca  $[p_1, p_2, p_3]$ . Esto visto como un filtro se ve en la figura 5.



(a) Shift register



(b) Diagrama de estados

Figura 6: Resolución de parte a.1 del ejercicio 9.6

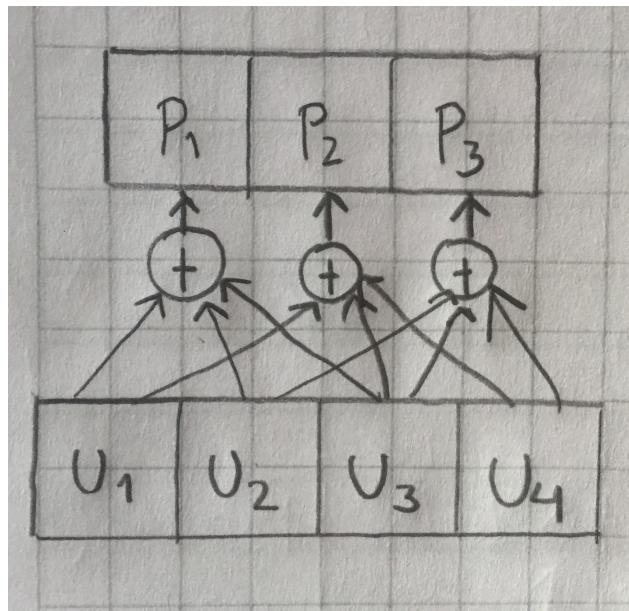
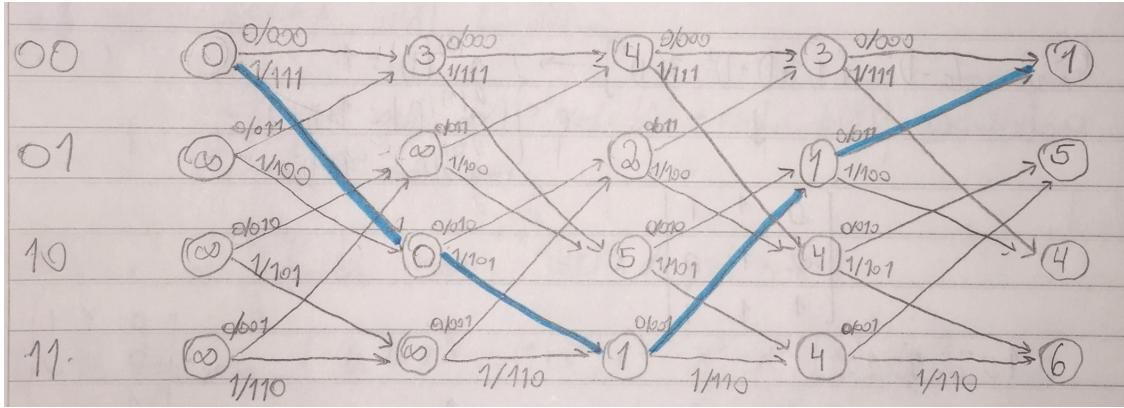


Figura 5: Implementación como filtro siguiendo idea presentada en [1]

#### 4.2. Ejercicio 9.6

En la figura 6a se adjunta la implementación del código convolucional como shift register. Para eso se constata la necesidad de existencia de dos buffers que permitan representar  $D$  y  $D^2$ . En la figura 6b se adjunta la representación del mismo en un diagrama de estados cuya entrada es el bit a la entrada y salidas son los tres bits de paridad.

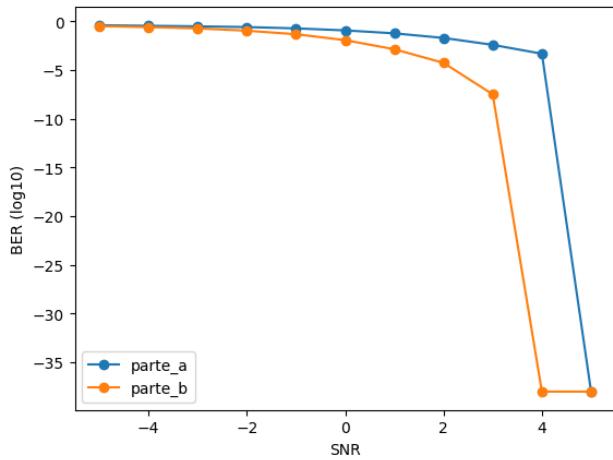
En la figura 7 se adjunta el diagrama de Trellis correspondiente. En el mismo tambien se adjunta el desarrollo del algoritmo de Viterbi para la entrada anotada arriba del diagrama. En azul se adjunta el camino que minimiza la distancia. Se constata que se cometio un error y se puede corregir siendo la entrada 1100.



**Figura 7:** Diagrama de Trellis junto con algoritmo de Viterbi

### 4.3. Ejercicio 9.7

En la figura 8 se adjunta la comparación del BER para sistema con y sin corrección de errores. Se ve como al principio son similares pero luego el desempeño es mucho mejor con la corrección de los mismos.



**Figura 8:** BER en escala logarítmica en función de la SNR

Para la parte de escuchar un audio se constato que sin corregir errores se percibe ruido desde  $-6,5db$  mientras que con corrector desde  $-4,5db$  lo que evidencia una mayor robustez frente al ruido al introducir corrección de errores.

## Referencias

- [1] Andrea Goldsmith. *Wireless Communications*. Cambridge University Press, Cambridge, UK, 1st edition, 2005.