Una introducción a los códigos detectores y correctores de errores

Pablo Fernández Gallardo[1] y Omar Gil Álvarez[2]

[1]Universidad Autónoma de Madrid, España
[2]Universidad de la República, Montevideo, Uruguay

Índice General

1 Introducción

El objetivo de estas notas es presentar una breve introducción a algunos aspectos de la teoría de códigos detectores y correctores de errores. El lector que tenga este material en sus manos notará que hemos fracasado en eso de lograr la brevedad, pero somos optimistas y confiamos en que este texto aún valga como una primera presentación de estos temas. Los problemas que aquí trataremos tienen que ver con la transmisión de información. Específicamente, con la búsqueda de *fidelidad*, a través del diseño de estrategias que permitan detectar y corregir posibles errores en la transmisión.

En general, al enviar un mensaje, o transferir información por cualquier procedimiento, deseamos que el receptor del mensaje reciba lo mismo que el emisor envió; y si no es así, que sea capaz de darse cuenta de que hubo errores en la transmisión. Veremos que se puede ir incluso más allá de la simple detección de errores, para dar al receptor la capacidad de corregirlso y recuperar el mensaje original. Es en este contexto donde desempeñan su papel los llamados códigos correctores de errores[3].

No nos detendremos en esta introducción a analizar en profundidad la importancia que tiene este tema en nuestra sociedad, porque parece innecesario hacerlo en momentos en que escuchamos permanentemente hablar de "la sociedad de la información", "banda ancha", etc, pero citemos algunas aplicaciones posibles:

- en las conversaciones telefónicas convencionales, donde el canal de transmisión es el cable, todos hemos experimentado en alguna ocasión las dificultades que ocasionan las interferencias de otras conversaciones o sonidos.

- Y también (¿más a menudo?) en la telefonía móvil: aquí, el canal es el espacio.

En general, los códigos que estudiaremos se hacen necesarios cuando nuestro canal es poco fiable, esto es, cuando tiene cierto *ruido*. El ruido puede ser de distintos tipos y responder a diversas causas, pero, esencialmente, su presencia implica que hay una cierta probabilidad de que el mensaje llegue alterado. En el caso particular de la telefonía móvil el ruido puede provenir, por ejemplo, de interferencias climatológicas.

- Cuando leemos o grabamos datos en el disco duro de un ordenador (o en un disco compacto), los dispositivos correspondientes (un láser, los cabezales) bien pueden cometer errores. El "ruido" aquí podría ser, por ejemplo, una huella dactilar en la superficie del CD.

[3] No abordaremos en estas notas otros problemas relativos a la transmisión de información, como son el de la seguridad y protección de información —que nos llevaría al ámbito de la *criptografía*—, o de la rapidez en la transmisión —vinculado con la *compresión* de la información—.

- En los supermercados existen aparatos dedicados a la lectura de los datos (códigos de barras, por ejemplo) que identifican los distintos productos. Estos códigos de barras están diseñados para detectar posibles errores de lectura.

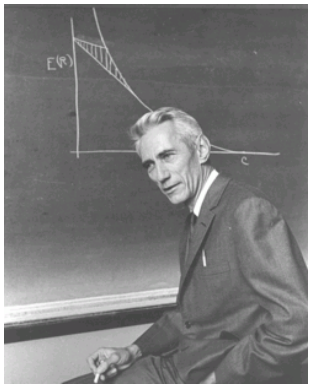
También asumiremos sin dar mayor detalle ni justificación que la información que transmitimos será siempre digital, esto es, largas listas de ceros y unos. La transcripción al lenguaje binario requerirá establecer un estándar adecuado: por ejemplo, el sistema ASCII y otros estándares de codificación relacionados con ASCII[4] convierten los símbolos habituales (letras del alfabeto, signos de puntuación) en números enteros (que no son sino listas de ceros y unos si los escribimos en base 2)[5], pero obviaremos estos problemas y tomaremos secuencias de ceros y unos como nuestro objeto de trabajo. En cualquier caso, la posibilidad de digitalizar cualquier información (música, películas, voz, fotografías, el código genético o lo que se nos ocurra) ya forma parte de las ideas corrientes y comúnmente aceptadas, por lo que también aquí ahorraremos explicaciones[6].

[4]Ver, por ejemplo, la tabla que corresponde al estándar ISO/IEC 8859-1 en el apéndice A.

[5]Esta transcripción tampoco es tan novedosa: el alfabeto Morse, que fue creado en 1838 por Samuel Morse, consiste precisamente en eso: convertir símbolos del idioma inglés en listas de ceros y unos (bueno, de puntos y rayas, sonidos cortos y largos, como lo queramos entender). Sin embargo, tuvo que pasar más de un siglo para que el impacto de esta idea, que apenas concebida permitió implementar las comunicaciones telegráficas, alcanzara todas las esferas de nuestra sociedad. Ver, al respecto, la nota 6.

Una breve historia (en inglés) sobre la evolución de los estándares para la codificación de caracteres puede encontrarse en `tronweb.super-nova.co.jp/characcodehist.html`. Incluye más información sobre el alfabeto morse, y el código ASCII y sus extensiones.

[6] Conviene señalar que, pese a que ya forma parte de nuestra manera de pensar, es una idea relativamente reciente, de mediados del siglo XX.



En 1948, Claude Shannon (1916-2001) publicó *A Mathematical Theory of Communication*, obra en la que estableció las bases de la moderna teoría de la información; fue el nacimiento de la era digital en la que hoy vivimos (y de las teorías de compresión de información o detección y corrección de errores que aquí trataremos). En esencia, Shannon concebía la información (de cualquier tipo) como un objeto matemático, lo que permitía *medir* la cantidad de información que contiene un mensaje, independientemente de su naturaleza. Intentemos explicarlo con un ejemplo: la frase "Brasil clasificó para la copa del mundo" contiene una cierta cantidad de información (vital, seguramente, para casi todos los brasileros); pero mucha menos, sin duda, por sorprendente e inesperado, de la que contendría la noticia de que fuera Uruguay el que logró la clasificación. Esta idea intuitiva —que podemos expresar en palabras comunes diciendo que la información que aporta un mensaje no depende tanto *de lo que diga*, sino de lo *inesperado* (improbable) que resulte— de que un mensaje puede

Lo que ya no es tan corriente es pensar que podamos definir operaciones entre listas de ceros y unos, y emplear algunas ideas geométricas sencillas para estudiarlas.

Introduciremos en la sección 3.1.2 el álgebra necesaria para operar con listas de ceros y unos. Notaremos algunas diferencias con las operaciones corrientes. Por ejemplo, tendremos que $1 + 1 = 0$. Pero muchas de las propiedades conocidas de las operaciones de suma y producto seguirán siendo válidas en este nuevo contexto. Veremos además que el conjunto de las listas de ceros y unos de una longitud dada admite una estructura de *espacio vectorial*, que resultará de enorme utilidad. Las consideraremos entonces como *vectores* que pueden ser sumados entre sí y multiplicados por escalares, de la misma manera que un par de números reales (x_1, x_2), o una terna de números reales (x_1, x_2, x_3), puede interpretarse como un vector en el plano o el espacio[7]. Sin embargo, el conjunto de escalares que utilizaremos será algo curioso, porque en vez de ser $\mathbb{R}$ se reduce al $\{0, 1\}$, y, a diferencia de los espacios $\mathbb{R}^2$ y $\mathbb{R}^3$, el conjunto formado por todos los vectores del espacio es finito[8].

Toda esta estructura lineal nos permitirá "ordenar" la información digital que manejaremos. De hecho, las ideas generales acerca de códigos correctores —introducción de redundancia, descodificación usando el criterio del vecino más próximo, etc.— que expondremos en la secciones 2 y 4 serían de poca utilidad sin la estructura lineal que permite implementarlas en algoritmos eficaces y aplicables a problemas reales.

Por otra parte, la noción de espacio vectorial lleva asociada consigo un conjunto de imágenes geométricas que sirven de guía a la intuición. El otro ingrediente que ayudará a construir una imagen geométrica de los problemas que trataremos es la introducción, en la sección 3.1.1, de una función *distancia* que medirá cuán separadas entre sí están las listas de ceros y unos. Para finalizar estos comentarios, deseamos subrayar que, en general, se ganan mucha claridad y espacio para la intuición cuando se introduce en cualquier dominio una interpretación geométrica adecuada. El conjunto de listas de ceros y unos de longitud n, o n-uplas de ceros y unos, no es una excepción. Y en esta situación que, en una primera mirada, cae fuera del ámbito de "la geometría" de rectas, planos y triángulos que nos acompaña desde los primeros años de la escuela, la visión geométrica también resulta fructífera.

contener más o menos información recibió una formulación matemática precisa a partir de los trabajos de Shannon.

[7]Una vez que se ha desarrollado esta intuición geométrica para los pares o ternas de números reales que forman los espacios $\mathbb{R}^2$ y $\mathbb{R}^3$ no es muy difícil dar un paso más y extenderla al espacio $\mathbb{R}^n$ formado por las n-uplas de reales, que, usando el lenguaje que empleamos en esta sección, no es otra cosa que el conjunto de listas de longitud n formadas por números reales.

[8]En efecto, el espacio vectorial formado por las listas de ceros y uno de longitud n tiene exactamente 2^n vectores, pero ya nos ocuparemos de estos detalles más adelante.

Este material esta extractado y adaptado del texto en preparación *Matemática Discreta*, de Fernandez&Fernández[9]. El primer autor quiere agradecer a Fernández&Fernández por permitir esta adaptación de algunas partes de su libro.

[9]José Luis Fernández y Pablo Fernández, de la Universidad Autónoma de Madrid.

2 Caracteres de control y aritmética módulo p.

Empecemos por presentar varios ejemplos de la vida cotidiana en los que podemos ver la función detectora y correctora de errores que queremos explicar. Nuestro primer ejemplo está alejado de las matemáticas, pero ilustra las ideas en juego.

Ejemplo 2.1 ALFABETOS TELEFÓNICOS
En muchas situaciones se hace necesario transmitir por teléfono o por radio una serie de letras. La más corriente es la que sufren periódicamente aquellas personas que tienen un apellido poco común e intentan deletrear su mombre cuando tienen que enviar sus datos por teléfono. En esas circunstancias a veces ocurren este tipo de conversaciones:

- Entonces es con "p".
- No, con "b" no. Con "p".
- Pero creo que eso le he dicho yo, ¿no? A ver, ¿es con "b" de "banana" o "p" de "perro"?
- Con "p" de "perro".

Y la cosa termina por ahí, a menos que el interlocutor entienda que es con "b" de "berro" y toda la confusión vuelva a empezar.

En un ambiente en el que es común transmitir secuencias de letras (por ejemplo un banco, una aerolínea, etc.) es usual adoptar como convención un "alfabeto telefónico" que acaba con estas ambigüedades. Uno bastante corriente en nuestro país está basado en nombres de mujeres[10]:

A	Alicia	**G**	Guillermina
B	Beatriz	**H**	Hombre[11]
C	Carolina	**I**	Inés
D	Dorotea	⋮	⋮
E	Eva	**Y**	Yolanda
F	Francisca	**Z**	Zapatería

Es fácil confundir "D" con "T", pero quien escuche "Torotea" en la transmisión inmediatamente sabrá que se trata de una "D". Vemos entonces que este mecanismo permite detectar errores en la transmisión y corregirlos. Hay muchas otras posibilidades para hacer esto. Por ejemplo, es usual

[10] En este alfabeto la letra "L" se codifica como "Lucía". Un funcionario bancario que habitualmente transmitía transacciones telefónicas contó a uno de los autores el siguiente hecho: algunos días después del 27 de junio de 1973 recibió en su agencia de Montevideo una transferencia desde otra ciudad. El mensaje correspondiente terminaba con una "L". Esta última "L" fue transmitida como "Libertad".

[11] ¿Por qué no "Hortensia"?

emplear en las compañías aéreas nombres de países y/o ciudades. También está extendido el "Alfa, Bravo, Charlie, ... Zulú", que suele escucharse en nuestra televisión. Por ejemplo, en cualquier serie o película de sábado por la tarde en que una avioneta esté a punto de hacer un aterrizaje forzoso en medio del desierto de Arizona o algún otro lugar inhóspito. ♣

Ejemplo 2.2 EL DÍGITO DE CONTROL DE LA CÉDULA DE IDENTIDAD URUGUAYA
La cédula de identidad uruguaya tiene un número de siete cifras, seguido de un dígito de una cifra. Éste es un carácter de control que se calcula tomando como base para el cálculo el número 2.987.634, al que llamaremos módulo verificador, e indicaremos con **m**. Para explicar el cálculo del dígito que corresponde a una cédula de identidad cualquiera tomaremos como ejemplo la cédula cuyo número es 1.913.577. Se procede de la siguiente manera: se multiplica la cifra de las unidades de **c**, por 4, que es la cifra de las unidades de **m**, y se guardan sólo las unidades. En nuestro ejemplo debemos hacer

$$7 \times 4 = 28 \mapsto 8.$$

El cálculo se repite con las cifras de las decenas. La cifra de las decenas de **c** se multiplica por la cifra de las decenas de **d**, es decir 3, pero sólo se conservan las unidades. En el caso particular que estamos considerando resulta

$$7 \times 3 = 21 \mapsto 1.$$

Repetimos esta operación dígito a dígito, multiplicando cada cifra del número de cédula por la correspondiente del módulo verificador, y conservando sólo las unidades. Los resultados para el 1.913.577 aparecen en la tabla 1 El siguiente paso es sumar todas las cifras correspondientes a las unidades

c	**m**	productos	unidades
1	2	2	2
9	9	81	1
1	8	8	8
3	7	21	1
5	6	30	0
7	3	21	1
7	4	28	8

Tabla 1: Cálculo de un dígito de control

que aparecen en la columna de la derecha, y, una vez más, conservar sólo las unidades del resultado:

$$2 + 1 + 8 + 1 + 0 + 1 + 8 = 21 \mapsto 1.$$

Por último, restamos de 10 la cifra a la que hemos llegado, repetimos el procedimiento de quedarnos con la cifra de las unidades y el resultado es el dígito de control de la cédula de identidad. En nuestro caso el resultado final es

$$10 - 1 = 9 \mapsto 9,$$

y a la cédula del ejemplo se le asigna el número 1.913.577-9. Si el número fuera 1.569.683 obtendríamos como dígito de control el 0.

Este dígito añadido es un ejemplo de lo que se llama un carácter de control porque permite detectar errores en la transmisión. Supongamos que intentamos transmitir el número de cédula del ejemplo y se produce un error como en el esquema

$$1.913.577\text{-}9 \quad \overset{\text{transmitimos}}{\longrightarrow} \quad 1.914.577\text{-}9.$$

Podemos detectarlo, porque al número 1.914.577 le corresponde el dígito 2. También podemos detectar el intercambio de dos cifras consecutivas, al que llamaremos *trabucazo*. Si cometiéramos un trabucazo entre la tercera y cuarta cifra obtendríamos 1.931.577-9. Notamos que hay un error, porque ahora el dígito de control debería ser 7.

Ejercicio 2.3 Completar los cálculos del dígito de control de los números

$$1.569.683, \quad 1.914.577, \quad \text{y} \quad 1.931.577.$$

Todavía más, si una de las cifras se perdiera en la transmisión la presencia del dígito de control permite recuperarla. En efecto, si se hubiera perdido el sexto dígito y recibiéramos 1.913.5X7-9, podemos calcular todas las filas de la tabla 1, salvo la sexta. Sumamos las unidades que aparecen en la última columna para estas seis filas y obtenemos 20. Como el dígito de control es un 9 sabemos que al sumar *todas* las unidades, incluyendo la que corresponde al dígito que se ha borrado, debemos obtener un 1 en la cifra de las unidades. Por lo tanto no está faltando un 1 en el sexto lugar. Ahora, ¿qué número entre 0 y 9 produce al ser multiplicado por 3 (el sexto dígito del módulo verificador $\mathbf{m}$) un resultado que tiene 1 en la cifra de las unidades? Repasemos la tabla del 3. Hay sólo una posibilidad para conseguir este 1: poner un 7 en el sexto lugar. Observemos que hemos recuperado correctamente el número de cédula original.

En el próximo ejercicio vamos a analizar más detalladamente las propiedades del dígito de control, y ver, por medio del estudio de algunos ejemplos, algunas de sus limitaciones.

Ejercicio 2.4

1. Calcular los dígitos de control de las cédulas 1.074.512, 1.073.512 y 1.074.012. ¿Qué comentario le merece el hecho de que los dígitos de control de dos de los números sean iguales?

2. En un documento encontramos el número de cédula 2.512.1X3-2, donde X representa un dígito ilegible. ¿Cuál es el dígito que falta? Y si el número fuera 2.5X5.555-3, ¿cómo habría que completarlo?
3. Mostrar que el dígito de control de la cédula de identidad permite detectar cualquier trabucazo (trasposición de dos dígitos consecutivos).
4. Sin embargo, si se intercambian cifras no consecutivas podría ocurrir que el error no quedara en evidencia. Dar un ejemplo de esta situación ♣

Los cálculos del último ejemplo utilizan sólo la cifra de las unidades de todos los números que aparecen. Notemos que para un número natural n cualquiera esta cifra es el resto de la división entera entre 10. En efecto, si

$$n = 10 \times q + r,$$

con $r < 10$, entonces la cifra de las unidades de n es r. Introduciremos a continuación la noción de *operación módulo un número natural* p, que está basada en los restos de la división entera entre p. En el caso particular $p = 10$ obtenemos la aritmética que empleamos en el ejemplo.

2.1 La aritmética módulo p

En el conjunto de los números naturales, calcular módulo un natural p cualquiera es hacer las siguientes operaciones: se suma –o multiplica– como siempre, pero en vez de dejar el resultado de esta operación tal como quedó se le toma como dividendo de una división entera con divisor p. Finalmente, nos quedamos con el resto de esta división como resultado final de nuestra suma –o producto–.

Ejemplo 2.5 Vamos a hacer algunos cálculos módulo 9. Si sumamos $22+7$ obtenemos

$$22 + 7 = 29 = 9 \times 3 + \mathbf{2},$$

de modo que el resultado de sumar 22 y 7 módulo 9 es 2. En realidad, como

$$22 = 9 \times 2 + \mathbf{4}$$

a los efectos del cálculo módulo 9 da lo mismo operar con 22 que con 4. En efecto,

$$4 + 7 = 11 = 9 \times 1 + \mathbf{2}$$

que también arroja resultado 2 si operamos módulo 9. En general, todos los múltiplos de 9 "desaparecen" en él cálculo módulo 9, y esté cálculo identifica entre sí —en el sentido de que no distingue entre ellos— dos números cualesquiera que difieran en un múltiplo de 9 (lo mismo es cierto respecto a los múltiplos de p en el cálculo módulo p, para cualquier valor natural de p). Veamos lo que ocurre con la multiplicación. Calculamos

$$22 \times 7 = 154 = 9 \times 17 + \mathbf{1},$$

y encontramos que el producto entre 22 y 7 es, módulo 9, igual a 1. Por supuesto, si en vez de usar el 22 operamos con cuatro obtenemos

$$4 \times 7 = 28 = 9 \times 3 + \mathbf{1},$$

que arroja el mismo resultado. ♣

Vemos entonces que en la aritmética módulo 9 el número 22 queda identificado con el 3, el 11 con el 2, etcétera. En general, cada número queda identificado con el resto que resulta de hacer su división entera entre 9. Así

$$100 = 9 \times 11 + \mathbf{1}$$

resulta equivalente a 1, y diremos que 1 es el **resto de** 100 **módulo** 9, lo que indicaremos también con

$$100 \equiv 1 \pmod 9$$

que refleja que 100 y 1 son indistinguibles para el cálculo módulo 9. Con la notación que acabamos de introducir escribiremos $23 \equiv 5 \pmod 9$, o diremos que 5 es el resto de 23 módulo 9.

Observemos que el resto módulo 9 de cualquier número entero es otro entero entre 0 y 8. En general, la división entera entre p sólo puede arrojar restos que estén en el conjunto $\{0, 1, \ldots, p-1\}$ formado por el 0 y los primeros $p-1$ naturales. Podemos considerar entonces que la aritmética módulo p que describimos actúa sobre el conjunto $\{0, 1, \ldots, p-1\}$ En estas notas adoptaremos este punto de vista[12], pero a la hora de calcular es muchas veces conveniente emplear el conjunto de todos los enteros, y usar el hecho de que cualquier múltiplo de p actúa como 0 cada vez que resulte conveniente para simplificar los cuentas.

Ejercicio 2.6 Calcular

$$53 \times (19 \times 3 + 5 + 12 \times (1 + 8))$$

módulo 7. Hacerlo de dos formas:

1. operando de la forma usual, y buscando recién al final del cálculo el resto módulo 7;

2. comenzando por sustituir 53, 19, 12 y 8 por sus restos módulo 7, y sustituyendo sistemáticamente todos los números que aparezcan en los cálculos por su resto número 7.

[12] Otra posibilidad es considerar que está definida sobre un conjunto de clases de equivalencia, en las que hemos identificado entre sí a todos los números que difieren en un múltiplo de p.

La suma y el producto módulo p comparten unas cuantas propiedades con las operaciones usuales. Una de ellas es que el 0 es neutro para la suma, ya que en la aritmética módulo p se satisface que $n + 0 = n$ para cualquier número n. También, todo número n tiene un opuesto que sumado a él da 0. En efecto, como observábamos antes podemos limitarnos a considerar $0 \leq n \leq p - 1$, y entonces $p - n$ es el opuesto de n, porque

$$n + (p - n) = p \equiv 0 \pmod{p}\,.$$

En la aritmética módulo 9 el opuesto de 4 es 5, pero es 6 en la aritmética módulo 10. En el ejercicio 2.8 presentamos y discutimos otras propiedades.

Ejemplo 2.7 UNA VEZ MÁS EL DÍGITO DE LA CÉDULA DE IDENTIDAD
El dígito de control de la cédula de identidad uruguaya puede describirse de manera muy breve haciendo referencia a las operaciones módulo 10. Se calcula de la siguiente manera: hay que multiplicar cada dígito dél número de cédula por el dígito correspondiente del número de control, luego sumar los resultados obtenidos y calcular el opuesto. Por supuesto, pero no olvidemos decirlo: todas estas operaciones deben hacerse módulo 10. ♣

Ejercicio 2.8 LAS PROPIEDADES DE LA ARITMÉTICA MÓDULO p
Designemos con los símbolos $+$ y $\cdot$ las operaciones de suma y producto de la aritmética módulo p. Con las letras a, b y c designaremos enteros cualesquiera en el conjunto $\{0, 1, \ldots, p-1\}$. Mostrar que se satisfacen las siguientes propiedades:

$a + b = b + a$	Conmutativa de $+$
$a + (b + c) = (a + b) + c$	Asociativa de $+$
$a + 0 = 0 + a = a$	Existencia de un elemento neutro para $+$
$\forall\ a\ \exists\ b$ tal que $a + b = b + a = 0$	Existencia de opuesto para $+$
$a \cdot b = b \cdot a$	Conmutativa de $\cdot$
$a \cdot (b \cdot c) = (a \cdot b) \cdot c$	Asociativa de $\cdot$
$a \cdot 1 = 1 \cdot a = a$	Existencia de un elemento neutro para $\cdot$
$a \cdot (b + c) = a \cdot b + a \cdot c$	Distributiva

El contenido de este ejercicio puede resumirse diciendo que el conjunto

$$\{0, 1, \ldots, p-1\}$$

con las operaciones de suma y producto forma un **anillo conmutativo con unidad**.

Ejemplo 2.9 EL CUERPO $\mathbb{F}_2$
El caso particular de la aritmética módulo p que se obtiene tomando $p = 2$ nos interesará especialmente en el futuro. Para este valor de p la aritmética opera sobre el conjunto

$$\{0, 1\}$$

que es la base de la codificación digital. Para $p = 2$ es completamente directo verificar que la suma y la multiplicación, actúan sobre los elementos de $\{0, 1\}$ según las siguientes tablas:

$+$	0	1
0	0	1
1	1	0

$\cdot$	0	1
0	0	0
1	0	1

En este caso vale una propiedad adicional que no es cierta para cualquier valor de p, y que consiste en que todos los elementos no nulos de $\{0, 1\}$ tienen inverso para la multiplicación. La verificación es obvia, porque el único elemento no nulo en ese conjunto es el 1, cuyo inverso es el propio 1.

Un conjunto en el que se definen dos operaciones binarias de manera que se verifican todas las propiedades recogidas en el ejercicio 2.8, más la existencia de inversos de todos los elementos no nulos se denomina un **cuerpo**. Es corriente designar[13] con el símbolo $\mathbb{F}_2$ al cuerpo formado por el conjunto $\{0, 1\}$ dotado de estas operaciones.

Ejercicio 2.10 Construir las tablas de suma y multiplicación para $p = 3$ y $p = 4$. Determinar los inversos de los elementos que lo tengan. ¿Cuál de los dos valores de p da origen a un cuerpo y cuál no? ♣

El próximo ejercicio muestra que la aritmética módulo p no está tan lejos de nuestra experiencia cotidiana.

Ejercicio 2.11 Calcular la tabla de la suma para $p = 12$. Sugerencia: recordar que todos sabemos que 7 horas despúes de las 8 de la mañana son las 3 de la tarde, y 11 horas más tarde las 2 de la mañana, etcétera.

2.2 Caracteres de control

Ya hemos visto cómo el dígito de control de la cédula de identidad puede describirse muy fácilmente empleando el lenguaje de la aritmética módulo p (con $p = 10$). En esta sección mostraremos cómo se construyen otros caracteres de control que el lector seguramente conocerá.

Ejemplo 2.12 CÓDIGOS DE BARRAS EN LOS ARTÍCULOS DE CONSUMO CORRIENTE
La mayor parte de los productos que consumimos a diario están identificados por medio de un número que aparece impreso en el exterior del envase y acompañado de un *código de barras* que hace posible su lectura por un escáner. Hay diversos estándares para esta codificación. A continuación

[13] El cambio de notación refleja el hecho de que al definir estas operaciones en el conjunto $\{0, 1\}$ se le ha añadido una estructura algebraica que lo convierte en un cuerpo (en inglés "field", de ahí la $\mathbb{F}$).

describiremos el más corriente en los comercios de Uruguay, que es conocido por el nombre de EAN-13. Cada producto aparece identificado por un número de trece cifras, que escribiremos en la forma

$$a_0a_1a_2a_3a_4a_5a_6a_7a_8a_9a_{10}a_{11}a_{12},$$

donde cada a_i, para $i = 0, 1, 2 \ldots, 12$, es un dígito entre 0 y 9. Los números $a_0a_1a_2a_3a_4a_5a_6$ contienen la identificación del fabricante[14], en tanto que el número de referencia del producto está formado por las cinco siguientes $a_7a_8a_9a_{10}a_{11}$. El dígito a_{12} se calcula de forma tal que

$$(a_0+a_2+a_4+a_6+a_8+a_{10}+\mathbf{a_{12}})+3(a_1+a_3+a_5+a_7+a_9+a_{11}) \equiv 0 \pmod{10}$$

Por ejemplo, los primeros doce dígitos del código que identifica a una cerveza son 8-411327-12201. Rápidamente calculamos

$$(8+1+3+7+2+0)+3(4+1+2+1+2+1) \equiv 4 \pmod{10},$$

por lo que el último dígito debe ser 6 y el número que identifica esta cerveza es 8-411327-122016.

Ejercicio 2.13

1. Calcular los dígitos de control para los siguientes números de producto:

   7-730406-00019, 7-730303-00021, 7-730241-00365.

2. Mostrar que éste sistema permite detectar los siguientes errores:

   (a) el cambio de una cifra (y sólo una) cualquiera del número de identificación del producto por otra;

   (b) la trasposición de dos cifras consecutivas, siempre que la diferencia entre ambas no sea igual a 5.

3. Introducir los siguientes "errores" en el número de la cerveza del ejemplo, de forma tal que el dígito de control no cambie:

[14] Hay dos organismos que actúan de forma coordinada a nivel mundial para asignar estos números de identificación y fijar los estándares de esta codificación de los productos. Estos son el UCC (Uniform Code Council) y EAN International (la sigla EAN significa European Article Number). A fines de los años 60 las industrias manufactureras y las cadenas de distribución norteamericanas reconocieron la necesidad de elaborar un sistema estandarizado de identificación de productos. Así nació el UPC (Universal Product Code) que tuvo una rápida expansión. Un proceso similar ocurrió en Europa, lo que originó la creación de un sistema llamado EAN que era compatible con el sistema norteamericano. La actual asociación EAN se formó en 1977, y cambió su nombre a EAN International en 1992, con el objetivo de reflejar la incorporación de países no europeos al sistema. Desde comienzos de los años 90 UCC y EAN han formalizado su colaboración para alcanzar un estándar prácticamente mundial en la codificación de productos. En el momento de preparación de este material mucha más información estaba disponible en las páginas web `www.ean-int.org` y `www.uc-council.org` de EAN y UCC.

(a) modificar exactamente dos cifras;

(b) intercambiar dos cifras consecutivas;

Vemos en este ejemplo de los códigos de identificación de productos la aplicación de algunas técnicas matemáticas a la resolución de un problema de indudable interés práctico[15]. ♣

El lector que haya analizado atentamente los ejemplos de la cédula de identidad uruguaya y de los códigos de barras habrá notado que el número 5 tiene la propiedad de introducir errores que los dígitos de control de esos ejemplos no pueden detectar. La clave es que el 5 es un divisor de 10, el número que hemos tomada como base para la aritmética módulo p con la que están construidos esos códigos. Usar una aritmética módulo 10 tiene la ventaja de que permite trabajar sobre el conjunto de los números $\{0, 1, \ldots, 9\}$, que coincide exactamente con los dígitos de nuestro sistema de numeración. Sin embargo, y tal como vimos, si se pretende usar 10 como base para la construcción de dígitos de control hay errores muy simples y fáciles de cometer, por ejemplo la sustitución de un dígito por otro, que no pueden ser detectados. Más eficientes son los códigos basados en un número primo. A continuación describimos con bastante detalle un ejemplo basado en el número 23, y mostramos que el problema de tener que usar "dígitos" más grandes que 9 no tiene mayor importancia[16].

Ejemplo 2.14 El número de identificación fiscal (NIF) español
El documento nacional de identidad (DNI) español consta de ocho cifras; para formar el NIF correspondiente, se le añade una letra al final. La elección de esta letra responde al siguiente procedimiento: sea n el número de 8 cifras que constituye el DNI. Se calcula el resto r de este número módulo 23,

$$n \equiv r \pmod{23},$$

con $0 \leq r \leq 22$. Con ese valor de r, se asigna una letra con la correspondencia que especifica la siguiente tabla:

0	1	2	3	4	5	6	7	8	9	10	11
T	R	W	A	G	M	Y	F	P	D	X	B

12	13	14	15	16	17	18	19	20	21	22
N	J	Z	S	Q	V	H	L	C	K	E

[15] Aunque no es el centro de nuestro interés, no perdamos de vista que la mayor dificultad en establecer un sistema uniforme de codificación de productos no está en las sencillas matemáticas de sus dígitos de control, sino en la creación y mantenimiento de un sistema de alcance prácticamente mundial. Al respecto ver, por ejemplo, la nota 14

[16] Este hecho es conocido, por ejemplo, para cualquier persona que esté familiarizada con la descripción de la paleta de colores de una computadora. El estándar es utilizar un sistema de numeración en base 16, en el que se emplean letras para representar los "dígitos" entre 10 y 15.

Obsérvese que no aparecen las letras O (se podría confundir con el 0), ni tampoco las letras I, U y Ñ. La letra así obtenida se añade al final del DNI para formar el NIF. Como vemos, este procedimiento no tiene ambigüedad posible: a cada DNI le corresponde una letra de la lista. Por ejemplo, al número 11809357 le corresponde la letra F. La elección del 23 no es casual: es un número primo y es casi el número de elementos de que consta el alfabeto castellano.

La letra final del NIF es otro ejemplo de un carácter de control que permite detectar errores de transmisión. Por ejemplo, supongamos que cometemos un error en la transmisión del DNI:

$$11809357\,\mathrm{F} \quad \overset{\text{transmitimos}}{\longrightarrow} \quad 21809357\,\mathrm{F}$$

Podemos detectar este error porque 7 no es el resto de 21809357 módulo 23 (que es lo que requeriría la letra F). Esto ocurre en general: este carácter extra detecta cualquier error (si es uno solo) en la transmisión, tal como mostraremos a continuación. Expresemos un número n de ocho cifras en su expresión decimal para analizar las propiedades de este dígito de control. Obtenemos

$$\begin{aligned} n &= n_7 n_6 n_5 n_4 n_3 n_2 n_1 n_0 \\ &= n_7\,10^7 + n_6\,10^6 + n_5\,10^5 + n_4\,10^4 + n_3\,10^3 + n_2\,10^2 + n_1\,10 + n_0 \\ &\equiv 14\,n_7 + 6\,n_6 + 19\,n_5 + 18\,n_4 + 11\,n_3 + 8\,n_2 + 10\,n_1 + n_0 \pmod{23}. \end{aligned}$$

La identidad en la fórmula está justificada porque $14, 6, 19, \ldots, 10, 1$ son los restos, módulo 23, de las distintas potencias de 10 involucradas. Observemos que todos esos coeficientes son distintos, y por supuesto, son primos[17] con 23. Supongamos que se comete un error en alguna posición:

$$n_7 \ldots n_j \ldots n_0 \quad \longrightarrow \quad n_7 \ldots \widetilde{n_j} \ldots n_0.$$

El primer número deja un resto r módulo 23, y el segundo, un resto digamos $\widetilde{r}$. El error pasaría inadvertido si estos dos restos fueran el mismo. Pero eso exigiría que

$$r = \widetilde{r} \quad \Longrightarrow \quad 10^j\,n_j \equiv 10^j\,\widetilde{n_j} \quad \Longrightarrow \quad n_j \equiv \widetilde{n_j} \pmod{23},$$

porque 10^j es primo con 23 para cualquier $j = 0, 1 \ldots, 7$. Y como tanto n_j como $\widetilde{n_j}$ son números menores que 9, ha de cumplirse que $n_j = \widetilde{n_j}$. Aunque este código detecta un error, no nos permite decidir en qué posición se ha producido, así que no vamos a poder corregirlo.

[17] Recordemos que dos números naturales son *primos entre sí* si el único divisor común a ambos es el 1.

Si se producen dos errores en la transmisión, en general no vamos a ser capaces de detectarlos. Por ejemplo, los números

$$11809357 \qquad \text{y} \qquad 11809311\,,$$

que se diferencian en las dos últimas cifras, devuelven resto 7 módulo 23, así que no podríamos detectar el error. Sin embargo, en el caso especial de los "trabucazos", en que se produce la trasposición de dos cifras, el código sí es útil. Supongamos que se comete un error de éstos:

$$n_7 \ldots n_j\, n_{j-1} \ldots n_0 \quad \longrightarrow \quad n_7 \ldots n_{j-1}\, n_j \ldots n_0\,.$$

Los números dejan restos r y $\widetilde{r}$, y el peligro sería que pudieran ser el mismo. Pero

$$\begin{aligned} r = \widetilde{r} \quad &\Longrightarrow \quad 10^j\, n_j + 10^{j-1}\, n_{j-1} \equiv 10^{j-1}\, n_j + 10^j\, n_{j-1} \pmod{23} \\ &\Longrightarrow \quad (n_{j-1} - n_j)\, 10^{j-1} \equiv (n_{j-1} - n_j)\, 10^j \pmod{23}\,. \end{aligned}$$

Como tanto n_j como n_{j-1} son enteros entre 0 y 9, su diferencia es un entero entre -9 y 9; además, $10^{j-1} \not\equiv 10^j$ módulo 23, como señalábamos antes. Así que sólo tendríamos que $r = \widetilde{r}$ si ocurriera que $n_j = n_{j-1}$, en cuyo caso la trasposición de ambos dígitos no supondría problema alguno.

Ejercicio 2.15 Mostrar que el dígito de control del NIF permite reconstruir un número al que le falta un dígito en una ubicación que conocemos. ♣

Ejemplo 2.16 El "International Standard Book Number" (ISBN)

La mayoría de los libros modernos están identificados por un número de 10 cifras $a_1a_2a_3a_4a_5a_6a_7a_8a_9a_{10}$ conocido como ISBN. Las nueve primeras cifras dan información sobre el libro (editorial, título, edición, etc.) y la última, a_{10}, es un caracter de control que se calcula para que

$$a_1 + 2a_2 + 3a_3 + 4a_4 + 5a_5 + 6a_6 + 7a_7 + 8a_8 + 9a_9 \equiv a_{10} \pmod{11}.$$

Como uno de los posibles restos de dividir entre 11 es 10 resulta que a_{10} podría tomar el valor 10. No parece deseable representar el número 10 de esta manera, ya que se confunde con el par de dígitos 1 0, de modo que para ello se utiliza la letra X (que es la forma tradicional de escribir 10 en el sistema de números romanos). Por ejemplo, existe el ISBN 84-239-5921-X (y corresponde a una enciclopedia).

Ejercicio 2.17 Mostrar que el carácter de control del ISBN permite detectar todos los errores en una cifra, puede recuperar un número cualquiera que se haya borrado (en ambos casos supondremos que hay exactamente un error) y el intercambio de dos cifras cualesquiera (consecutivas o no). ♣

Ejemplo 2.18 LA PRUEBA DEL NUEVE
La clásica prueba del nueve cae en el contexto de la aritmética módulo 9 y los caracteres de control que permiten detectar errores. Recordemos que esta prueba consiste en asociar a cada número natural x un dígito $N(x)$ que se calcula sumando las cifras de x, y repitiendo esta operación hasta obtener un número de una cifra. Para verificar el resultado de una operación de suma o producto entre naturales se realiza la misma operación con los dígitos $N(x)$. El próximo ejercicio tiene como objetivo justificar las buenas propiedades de la prueba del nueve.

Ejercicio 2.19

1. Probar que la diferencia entre un número natural x y la suma de sus cifras en la expresión decimal del número es un múltiplo de 9. Mostrar que si el número tiene más de una cifra entonces la suma de sus cifras es menor que el número.
2. Mostrar que si el procedimiento de sumar las cifras de un número natural x se itera[18] hasta obtener un número $N(x)$ de una sola cifra, entonces $N(x)$ es igual al resto de n módulo 9, salvo en el caso en que $N(x)$ es 9. Mostrar además que $N(x) = 9$ si y sólo si x es un múltiplo de 9.
3. Probar que
$$N(x + y) = N(N(x) + N(y)), \quad N(x \times y) = N(N(x) \times N(y)).$$
4. Usar la información de las partes anteriores del ejercicio para justificar completamente la prueba del 9. ♣

[18]Es decir, como primer paso se calcula la suma de las cifras del número. Si esta suma tiene más de una cifra sus cifras se suman, etcétera, tal como se hace al realizar la prueba del 9.

3 Códigos correctores y listas de ceros y unos

Los ejemplos de la sección anterior tienen en común el que tratamos de operar con un cierta lista de números y que añadimos un carácter extra para detectar posibles errores. Una idea fundamental que debemos extraer de ellos es la siguiente: *podemos detectar errores porque añadimos cierta información adicional* o *redundancia*. Pero no queremos restringirnos a la transmisión de números; también queremos enviar mensajes de otro tipo (por ejemplo, de texto), y queremos encontrar mecanismos que hagan la misma labor de detección de errores y aún más: *mostraremos como corregir algunos de los errores cometidos*[19]. Haremos todo esto a través de la elaboración de **códigos** diseñados en forma tal que detecten y corrijan errores.

Para empezar a discutir las características que debería tener un código hagamos algunas observaciones en un contexto general, todavía no matematizado. Imaginemos que recibimos el siguiente mensaje de texto:

"*En un lular de la Mancha*".

Incluso si nos olvidamos de que la frase es muy conocida, nos damos cuenta inmediatamente de que se han producido errores en la transmisión, porque "lular" no es una palabra del castellano. Ésta es una idea importante: *no todas las combinaciones de letras son palabras válidas de nuestro diccionario*; y esto nos permite detectar en este caso el error. Un código debería replicar esta propiedad de alguna manera. Es más, el contexto nos permite intuir que el error está en la segunda palabra, que debería ser "lugar". Pero al codificar esta frase con listas de ceros y unos perderemos, por supuesto, esta idea del contexto. De alguna manera, *un código debe generar un "contexto" que nos permita corregir.* ¿Y por qué decidimos que es "lugar" la palabra correcta? También podría haber sido "lunar", pero no parece adecuada al contexto. Pero sí sería adecuada la palabra "solar". Ahora bien, en este caso se deberían haber producido dos errores en la transmisión (en las dos primeras letras); y parece que esto es bastante más improbable.

¿Y si hubiéramos recibido

"*En un lpgar de la Mancha*"?

De nuevo detectaríamos un error, y estaríamos tentados de corregirlo de igual manera, con "lugar". Pero, ¿por qué no con "lagar"? Parecen igual de apropiadas, por contexto[20] y por número de errores cometidos. *Un código debería evitar estas situaciones de ambigüedad.* Sigamos con esta idea. Imaginemos ahora que recibimos la palabra "Maltonado". Cualquiera detectaría

[19] En los ejemplos 2.2, 2.14 y 2.16 de la sección 2 vimos un caso particular de esta función correctora: la recuperación de caracteres borrados.

[20] Un lagar es un recipiente donde se pisa la uva (también la manzana o la aceituna) para obtener el mosto (sidra o aceite). Por extensión, es el edificio donde hay un lagar.

y podría corregir el error cometido, con la palabra "Maldonado"; y podría hacerlo porque no hay palabras del castellano que estén "cerca" de "Maldonado". Sin embargo, si recibimos "hasa", nos resultaría difícil corregir el error: podría ser casa, masa, tasa, ... En general, cualquier proceso de codificación y descodificación (en particular, en su aplicación a la corrección de errores) deberá resolver estos problemas y hacer que la descodificación sea *única*. Esto es, el algoritmo que deshaga la codificación ha de devolvernos un único resultado.

Resumamos todas estas ideas:

- un código debería contener cierta información extra, que nos permita detectar y corregir errores.
- Nuestro diccionario no debería contener todas las palabras posibles (pero sí las suficientes).
- Y las palabras de que conste deberían estar "separadas" unas de otras en algún sentido.
- También es conveniente tener en cuenta en el diseño del código que tan fácil es que se produzcan errores. Para manejar este aspecto de la cuestión, esencialmente aleatorio, hay que recurrir a modelos en los que intervenga la probabilidad de cometer uno, dos, ... errores[21].

¿Cómo recoger todas estas propiedades en la elaboración de un código? Aproximarnos a una respuesta para esta pregunta será el objetivo del resto de este artículo. Las consideraciones que hemos hecho hasta ahora sugieren algunas características cualitativas para nuestros códigos, que buscaremos implementar. Para ello comenzaremos por escribir cualquier tipo de información de una manera que permita un tratamiento matemático del problema.

Asumiremos entonces que se trata de transmitir información que ha sido *digitalizada* en forma de una secuencia de ceros y unos. Esto no es difícil de hacer si la información es numérica, porque podemos representar los números en notación binaria, usando 2 como base del sistema de numeración en vez del sistema decimal habitual construido sobre el 10, lo que requiere sólo los dígitos 0 y 1. También es posible traducir caracteres de texto, imágenes, música, etc, en listas de ceros y unos. Para los textos el estándar más corriente es el fijado por el sistema ASCII. En este sistema los símbolos que se utilizan habitualmente en el idioma inglés (letras del alfabeto, signos de puntuación, etc.) están asociados a un número natural entre 0 y 127, que, escrito en notación binaria, es una lista de ceros y unos de longitud 7. Sin

[21] Este aspecto de la cuestión no será el centro de nuestra discusión, y nos bastará para esta presentación la idea de que es más difícil o improbable cometer dos errores que uno solo. En la sección 5 iremos un poco más lejos y analizaremos los códigos correctores incorporando algunos elementos del cálculo de probabilidades.

embargo, símbolos corrientes en el español, como las vocales acentuadas y la letra "ñ" quedan fuera de este código ASCII de "siete bits", por lo que emplearemos el estándar de codificación ISO/IEC 8859-1, también conocido como "Latin-1". Éste es una ampliación del código ASCII que emplea números entre 0 y 255 e incorpora los carácteres que se emplean en los idiomas del sur de Europa[22]. Como cada número natural comprendido entre 0 y 255 puede representarse por una lista de ceros y unos de longitud 8, ver nuestro próximo ejemplo, esta codificación permite traducir cualquier texto en secuencias de ceros y unos. Hemos incluido una tabla con la norma ISO/IEC 8859-1 (Latin-1) en el apéndice A.

Ejemplo 3.1 JUGANDO CON EL CÓDIGO ASCII Y LATIN-1
En este ejemplo veremos como opera la representación de un texto en forma de listas de ceros y unos. Queremos enviar la palabra "casa" (en minúsculas). Si usamos el estándar fijado por el código ASCII, la letra a viene representada por el número 97, la c por el 99 y la s, por el 115. Dado que

$$\begin{aligned} 97 &= \mathbf{1}\times 2^6+\mathbf{1}\times 2^5+\mathbf{0}\times 2^4+\mathbf{0}\times 2^3+\mathbf{0}\times 2^2+\mathbf{0}\times 2^1+\mathbf{1}\times 2^0,\\ 99 &= \mathbf{1}\times 2^6+\mathbf{1}\times 2^5+\mathbf{0}\times 2^4+\mathbf{0}\times 2^3+\mathbf{0}\times 2^2+\mathbf{1}\times 2^1+\mathbf{1}\times 2^0,\\ 115 &= \mathbf{1}\times 2^6+\mathbf{1}\times 2^5+\mathbf{1}\times 2^4+\mathbf{0}\times 2^3+\mathbf{0}\times 2^2+\mathbf{1}\times 2^1+\mathbf{1}\times 2^0, \end{aligned}$$

escribiendo en binario y en el orden adecuado, obtendríamos

1 1 0 0 0 1 1	1 1 0 0 0 0 1	1 1 1 0 0 1 1	1 1 0 0 0 0 1
c	a	s	a

Los mismos números corresponden a estas letras en Latin-1. Pero como este estándar codifica con ocho dígitos, hay que anteponer un cero delante de cada secuencia de siete dígitos para pasar de ASCII a Latin-1.

Ejercicio 3.2 Los números que correspondes a las letras "C" y "ñ" en Latin-1 son, respectivamente, 67 y 241. Hallar sus codificaciones binarias.

Ejercicio 3.3 Traducir, según Latin-1, en sucesiones de ceros y unos las viejas y queridas frases "Mi mamá me mima"y "Ese oso se asea".

Llamaremos **palabras**[23] a las listas de ceros y unos, de longitud digamos k, que podremos utilizar en un mensaje. Dispondremos, pues, de un

[22] La norma ISO/IEC 8859 contiene especificaciones para la codificación de símbolos de diversos lenguajes en un conjunto de tablas de 256 caracteres. La primera mitad de todas las tablas es igual, y coincide con la codificación ASCII. En la segunda mitad se introducen las variantes correspondientes a cada lenguaje, o familia de lenguajes específicos. Usaremos la primera tabla de esta norma, que contiene los símbolos necesarios para el idioma español.

[23] Conviene no despistarse con los términos utilizados: llamaremos palabras a cada lista del diccionario. Pero, en general, no serán palabras en el sentido usual: por ejemplo, en un sistema tipo ASCII, cada lista de éstas representará un símbolo, una letra.

conjunto de palabras o **diccionario** $\mathcal{P}$ que, en general, será una familia de listas de longitud k formadas por ceros y unos. Es usual llamar $\{0,1\}^k$ al conjunto de todas las listas de longitud k formadas por ceros y unos. En esta notación la letra k representa la longitud de las listas, en tanto que $\{0,1\}$ es el conjunto del que vamos a escoger los elementos de la lista[24]. Con esta notación, tenemos que el diccionario $\mathcal{P}$ es un cierto subconjunto de $\{0,1\}^k$, lo que expresamos con la fórmula

$$\mathcal{P} \subset \{0,1\}^k .$$

La inclusión puede ser estricta, lo que significa que algunas combinaciones de ceros y unos son palabras que no tienen sentido para nosotros porque no están en nuestro diccionario[25], o puede ocurrir que $\mathcal{P}$ sea todo el conjunto $\{0,1\}^k$. En casi todos los ejemplos que consideraremos estaremos en esta última situación.

En la sección anterior mostramos como era posible detectar errores empleando un dígito de control. Al poner esto en práctica teníamos que agregar a la información que ya teníamos (número de cédula de identidad, código de un producto, etc.) un carácter extra (letra o número). En un proceso de codificación general ocurre algo parecido. Una vez escrito el mensaje en los términos adecuados, el codificador transformará cada una de las palabras de longitud k del mensaje en otra lista de ceros y unos, que tendrá en general una longitud distinta, $n > k$. La información introducida en los $n-k$ dígitos que se agregan en el proceso de codificación crea la redundancia que nos permitirá detectar y corregir errores. Estas nuevas listas serán las **palabras código**, que formarán parte de otro diccionario $\mathcal{C}$, el **código**:

$$\mathcal{C} \subset \{0,1\}^n .$$

La inclusión de $\mathcal{C}$ en $\{0,1\}^n$ será estricta en todas las aplicaciones que tengan algún interés[26]. Como ya vimos, esta es una propiedad deseable para nuestro diccionario si pretendemos corregir errores.

La **codificación** es la receta f que nos permite pasar de las palabras a las palabras código, poniendo cada palabra $\mathbf{p}$ en correspondencia con una palabra código $\mathbf{c}$. Se trata entonces de una función que a cada lista $\mathbf{c}$ de longitud k que está en el diccionario $\mathcal{P}$ le asocia una lista $\mathbf{p}$ de n posiciones que está en el código $\mathcal{C}$. Escribiremos $\mathbf{p} = f(\mathbf{c})$ para indicar que $\mathbf{p}$ se obtiene

[24] En general, el conjunto X^m está formado por todas las listas de m elementos de X. Por ejemplo, llamamos $\mathbb{R}^n$ al conjunto de las n-uplas (listas de longitud n) de números reales.

[25] Esto es completamente análogo a lo que ocurre con el lenguaje habitual y la mayoría de las palabras que se obtienen tecleando al azar.

[26] En realidad, si $n > k$ e incluimos en el código sólo las palabras que resulten de codificar palabras de $\mathcal{P}$, un sencillo argumento de cardinalidad muestra que la inclusión tiene que ser estricta.

de $\mathbf{c}$ por medio de la función de codificación f. Toda la información de este parrafo está resumida en la siguiente notación:

$$\begin{aligned} f : \mathcal{P} \subset \{0,1\}^k &\rightarrow \mathcal{C} \subset \{0,1\}^n, \\ \mathbf{p} &\mapsto f(\mathbf{p}) = \mathbf{c}. \end{aligned}$$

La primera linea dice que f es una función definida sobre el diccionario $\mathcal{P}$ que está formado por listas de ceros y unos de longitud k, y que toma valores en el código $\mathcal{C}$, que está dentro del conjunto de las n-listas de ceros y unos. La segunda línea expresa el hecho de que f envía una palabra $\mathbf{p}$ a una palabra $f(\mathbf{p})$ que llamaremos $\mathbf{c}$. En fin, no es nada más ¡ni nada menos! que una notación. Puede ser un poco engorrosa al principio, pero expresa con mucha economía toda la información que necesitamos.

¿Cuál es la utilidad de todo esto? Cuando queramos transmitir $\mathbf{p}$ la codificaremos y enviaremos en su lugar $\mathbf{c}$. ¿Por qué? Porque si se producen errores en la transmisión sabremos corregirlos, al menos en algunos casos, a partir de la redundancia que el código introduce. No seríamos capaces de eliminar errores, ni siquiera de enterarnos de que un error se produjo, si enviáramos la palabra $\mathbf{p}$ sin procesar.

Subrayemos que en todo esto el código, aunque será nuestro principal objeto de estudio, cumple una función auxiliar que consiste en ayudarnos a transmitir de forma fiel las palabras del diccionario original $\mathcal{P}$, que son los objetos que realmente nos interesan. Por lo tanto, cada vez que quien actúe como receptor de un mensaje tenga una palabra $\mathbf{c}$ del código deseará saber cuál es la palabra $\mathbf{p}$ de la que proviene. Hay un requisito obvio que debe satisfacer entonces la función de codificación: *no puede enviar dos palabras diferentes de $\mathcal{P}$ a la misma palabra de $\mathcal{C}$*. Si esto ocurriera no podríamos descodificar. Por lo tanto, la función f tiene que ser *inyectiva*. El segundo requisito que impondremos es que *toda palabra del código $\mathcal{C}$ provenga de una palabra de $\mathcal{C}$ a través de la función de codificación*, cosa que permitirá determinar $\mathbf{p}$ para cualquier $\mathbf{c}$. Este segundo requisito implica que la función f debe ser *sobreyectiva*[27]. Estas dos condiciones aseguran que la codificación puede deshacerse para cualquier palabra $\mathbf{c} \in \mathcal{C}$. Naturalmente, esta descodificación de palabras código es equivalente a invertir la función f para obtener

$$f^{-1} : \mathcal{C} \rightarrow \mathcal{P}.$$

Por lo tanto, si se recibe $\mathbf{c}$ debemos interpretar que se está enviando la palabra $\mathbf{p} = f^{-1}(\mathbf{c})$ del diccionario $\mathcal{P}$.

[27]Este requisito es menos serio que el anterior. Si una función inyectiva f no lo cumple podemos modificar el código $\mathcal{C}$ desechando las palabras que no vienen de una palabra de $\mathcal{P}$, para quedarnos solamente con la que son de la forma $\mathbf{c} = f(\mathbf{p})$ para alguna palabra $\mathbf{p} \in \mathcal{P}$. En otros términos, definimos como nuestro nuevo código la imagen de $\mathcal{P}$ por la función f, lo que automáticamente produce una función de codificación biyectiva.

El diseño de buenos códigos depende de una elección sabia de f, lo que incluye, entre otras cosas, fijar los valores de k y n. En el marco de las listas de ceros y unos tendremos una estructura geométrica de la que asirnos, y la posibilidad de operar algebraicamente con las palabras de nuestro diccionario. Podremos, en consecuencia, abordar el problema del diseño de códigos en ese contexto. Nuestro próximo paso será, precisamente, estudiar *el espacio de las listas de ceros y unos.*

3.1 Las listas de ceros y unos

3.1.1 Su Geometría

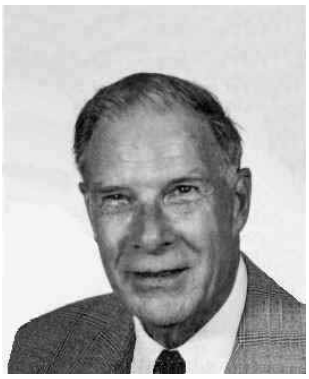
Comencemos con un sencillo ejemplo: supongamos que tenemos tres listas de ceros y unos de longitud 5,

1	0	0	0	1		1	1	0	0	1		0	0	1	1	1

Cualquier persona diría que las dos primeras se parecen entre sí, que están mas "cerca" entre ellas que con respecto a la tercera. Estamos aplicando un criterio natural para compararlas: el número de posiciones en que las listas difieren. Así, las dos primeras sólo difieren en la segunda posición, mientras que, en relación a la tercera, difieren en más posiciones. Lo que vamos a hacer es dar nombre a esta forma de medir lo "distintas" que son dos listas de éstas. Veremos que la manera de hacerlo es mediante una distancia, así que la tentación primera de usar las palabras "cerca" o "lejos" resulta ser bastante acertada.

Definición 1 *Consideremos el conjunto de todas las n-listas formadas con ceros y unos. Entonces, definimos la* ***distancia de Hamming***[28] *entre dos*

[28] Richard Hamming (1915–1998) es uno de los héroes de esta historia. Trabajaba, al igual que Shannon, ver la nota 6 en la página 4, en los laboratorios de la compañía telefónica Bell.



Cuenta la leyenda que su interés por diseñar códigos correctores de errores se despertó por razones prácticas: las computadoras de entonces trabajaban con fichas perforadas, y los usuarios escribían (con notable esfuerzo) sus programas sobre ellas. El paquete de fichas se entregaba al operario, que las introducía en la computadora para su compilación. Si se detectaba algún error (lo que era bastante frecuente) en alguna ficha, la computadora se detenía. Eso era especialmente dramático en los fines de semana: el trabajo se interrumpía y se pasaba a procesar el siguiente. El investigador, que había dedicado parte de la semana a escribir las fichas, descubría consternado que apenas se habían compilado un par de ellas. Pero si la computadora es capaz de detectar esos errores, pensaba Hamming, ¿no habrá alguna manera de que también los corrija? La necesidad aguza el ingenio, sin duda.

listas $\mathbf{x} = (x_1, \ldots, x_n)$ *e* $\mathbf{y} = (y_1, \ldots, y_n)$ *como el número de posiciones en que ambas difieren.*

Se puede comprobar que, efectivamente, esta distancia de Hamming es una distancia, en el sentido matemático del término. Como es la primera vez que aparece este concepto en el texto, lo definiremos adecuadamente: una función d que lee pares de elementos de un cierto conjunto E y devuelve números reales, es decir, una función

$$\begin{array}{rccc} d: & E \times E & \longrightarrow & \mathbb{R} \\ & (x, y) & \longmapsto & d(x, y) \end{array}$$

es una **distancia** o **métrica** si cumple las siguientes propiedades:

1. $d(x, y) \geq 0$ para todo par de elementos $x, y \in E$ (esto es, es una función que toma valores no negativos).

2. $d(x, y) = 0$ si y sólo si $x = y$ (si dos elementos distan cero entre sí es que son, en realidad, el mismo; y viceversa).

3. $d(x, y) = d(y, x)$ para cada par de elementos $x, y \in E$ (es una función simétrica).

4. $d(x, y) + d(y, z) \geq d(x, z)$ para cada tres elementos $x, y, z \in E$ (desigualdad triangular).

El conjunto E, junto con la función distancia d, se denomina **espacio métrico**. Por supuesto, la distancia habitual en el plano o en el espacio (la distancia euclídea a la que estamos tan acostumbrados) es una distancia en el sentido que acabamos de introducir[29]. En el caso que nos ocupa, el de la distancia de Hamming en el conjunto de las n-listas de ceros y unos, es fácil comprobar que cumple las propiedades expuestas anteriormente: las tres primeras propiedades son casi inmediatas de verificar. Sólo la desigualdad triangular requiere un momento de reflexión, que dejamos como ejercicio para el lector.

Ya que hemos introducido un concepto de distancia en el conjunto de las n-listas de ceros y unos, intentemos utilizar la intuición geométrica que tenemos en el plano o en el espacio. Por ejemplo, una "bola" en el plano centrada en un cierto punto $\mathbf{a}$ y de radio r, $B_r(\mathbf{a})$, es el conjunto de puntos que distan, a lo sumo, r del punto $\mathbf{a}$:

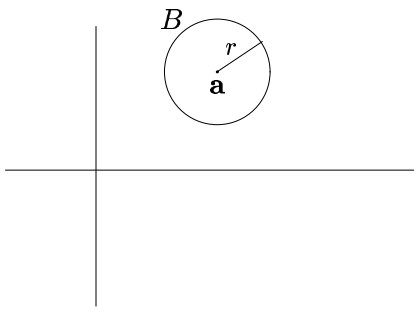
$$B_r(\mathbf{a}) = \{\mathbf{x} \in \mathbb{R}^2 \ : \ |\mathbf{x} - \mathbf{a}| \leq r\},$$

[29] En realidad la noción de distancia aparece en muchos contextos diferentes, y es justamente este hecho el que justifica el interés de introducir una definición general de distancia que recoja los aspectos comunes a ejemplos que pueden parecer, en una primera mirada, muy diferentes entre sí. En general, a cualquier conjunto E dotado de una función distancia d se le llama **espacio métrico.**

donde la distancia euclídea entre los puntos $\mathbf{a}$ y $\mathbf{x}$ es, por supuesto, el módulo del vector $\mathbf{x}-\mathbf{a}$. El mismo objeto geométrico se puede describir para nuestra distancia de Hamming: así, diremos que la bola de radio r en torno a la n-lista de ceros y unos $\mathbf{a}$ es el conjunto de n-listas que están a distancia menor o igual que r. Si usamos la notación $B_r(\mathbf{a})$ para designar a este conjunto, tenemos entonces

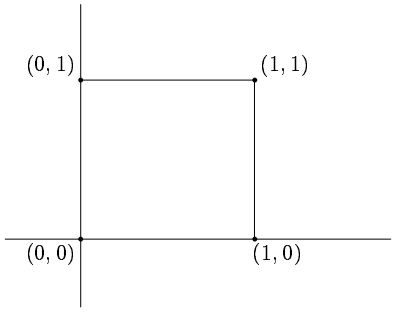
$$B_r(\mathbf{a}) = \{n\text{-listas } \mathbf{x} \,:\, d(\mathbf{x}, \mathbf{a}) \leq r\}\,,$$

donde d es la distancia de Hamming, el número de posiciones en que difieren dos listas[30]. Conviene diferenciar entre las descripciones gráficas que tienen estos objetos. Por ejemplo, la bola en $\mathbb{R}^2$ es el conjunto de puntos que viven dentro de un círculo centrado en $\mathbf{a}$ de radio r,

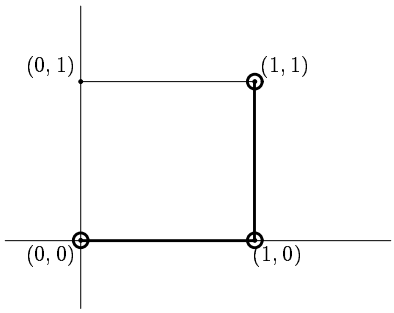


mientras que la bola en $\mathbb{R}^3$ es una esfera. Para el caso de las n-listas de ceros y unos, claro, la geometría no es la misma, aunque a veces nos será de ayuda pensar en las bolas habituales. Hay varias maneras de representar las bolas en este caso. Por ejemplo, para listas de dos posiciones, lo mejor es representar cada una de las cuatro listas, $(0,0)$, $(1,0)$, $(0,1)$ y $(1,1)$, como vértices de un cuadrado en el plano:

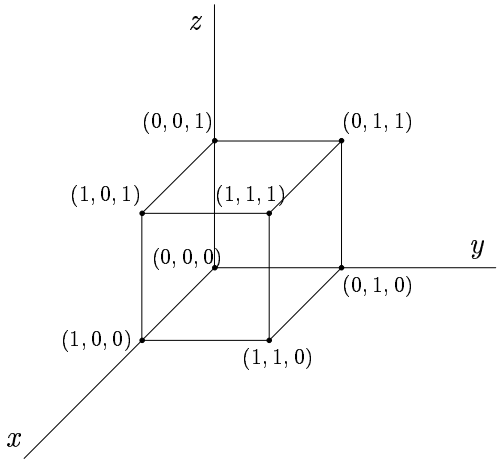
[30]Vale la pena mencionar que lo más corriente es definir la bola $B_r(\mathbf{a})$ en $\mathbb{R}^n$ como el conjunto formado por los x tales que $|x-a| < r$. La diferencia entre pedir que la igualdad se satisfaga en forma estricta ($<$) o no ($\leq$) es la diferencia entre considerar bolas *abiertas* (que no contienen a su frontera) o *cerradas* (que sí la contienen). Estas bolas tienen propiedades bastante diferentes en $\mathbb{R}^n$, pero en el espacio de listas que estamos considerando –un espacio discreto, donde los puntos distintos están a una distancia que es por lo menos igual a 1– la única diferencia es la siguiente: si definiéramos las bolas poniendo un menor estricto, entonces los puntos a distancia 1 de $\mathbf{a}$ no estarían en $B_1(\mathbf{a})$, y habría que tomar bolas con radio $r \in (1,2]$ para incluir los puntos a distancia 1 y dejar fuera los que están a distancia 2 o mayor. Con la definición que hemos dado la bola de radio 1 incluye las listas a distancia 1, y deja fuera las que están a distancias mayores. En general, la de radio m, con m natural, contiene a las que están a distancia m, pero no a las que distan $m+1$.



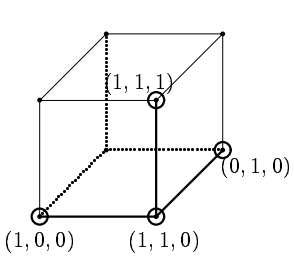
Dos listas de dos posiciones comparten un lado del cuadrado si difieren en una sola posición, así que la bola de radio 1 centrada, por ejemplo, en la lista $(1, 0)$ incluiría, además de a esta lista, a las dos que etiquetan vértices del cuadrado con los que comparte una arista:



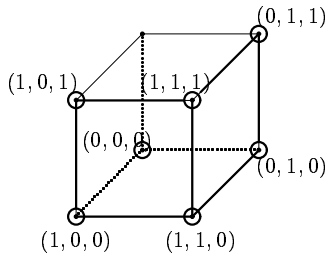
Y la bola de radio 2 ya incluiría a todas las listas del conjunto. Para las listas de tres posiciones con ceros y unos utilizaremos los vértices del cubo unidad en $\mathbb{R}^3$. De nuevo, dos listas diferirán en una posición (distarán una unidad con la distancia de Hamming) si comparten una arista del cubo:



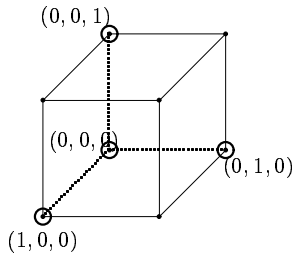
Con este esquema, es fácil darse cuenta de que una bola de radio 1 comprende, además de a la lista que es centro de la bola, a las tres con las que comparte aristas. Dibujamos a continuación varias bolas en esta geometría:



$B_1((1,1,0))$



$B_2((1,1,0))$



$B_1((0,0,0))$

Cualquier bola de radio tres incluye ya a todas las listas.

En $\mathbb{R}^n$ las bolas constan de un número infinito de puntos si el radio r es mayor que 0. En el contexto de las n-listas constarán de un número finito de elementos. Por ejemplo, ¿cuántas n-listas están a distancia menor o igual que 1 de una dada? Esto es, ¿de cuántos elementos consta una bola de radio 1 en esta geometría? Los dibujos exhibidos anteriormente nos permiten intuir que habrá $n+1$ elementos (el centro y n vecinos) en la bola de radio unidad, sea cual sea su centro. Podemos comprobarlo directamente: dada una lista de n posiciones con ceros y unos, las listas que estén a distancia 1 son las que difieren en una única posición, y hay n posibles posiciones que cambiar. ¿Y en la bola de radio 2? Habrá que contar el centro, las que estén a distancia 1 y las que estén a distancia 2. De estas últimas hay tantas como las maneras de elegir exactamente dos posiciones en las que introducir una modificación respecto a la palabra que está en el centro de la bola. Esta cantidad es $\binom{n}{2}$. En nuestro cálculo anterior el número n apareció al considerar todas las posibilidades para introducir *una* modificación en una lista de longitud n; observemos que $n = \binom{n}{1}$. En total, para una bola B_2 de radio 2 tenemos que la cantidad de elementos que contiene es[31]

$$|B_2| = \sum_{j=0}^{2} \binom{n}{j} = 1 + n + \frac{(n(n-1)}{2}.$$

En general, es posible obtener $\binom{n}{j}$ palabras modificando una lista dada de longitud n, formada por ceros y unos, en exactamente j posiciones. Por lo tanto, la bola de radio r en el espacio de las n-listas de ceros y unos tendrá

$$|B_r| = \sum_{j=0}^{r} \binom{n}{j}$$

[31] Estamos empleando aquí la notación $|X|$ para indicar la cantidad de elementos, o *cardinal*, de un conjunto X. Por ejemplo, con esta notación podemos escribir la distancia de Hamming entre dos listas $\mathbf{x}$ e $\mathbf{y}$ en la forma

$$d(\mathbf{x},\mathbf{y}) = |\{j \,:\, 1 \le j \le n\,,\, x_j \neq y_j\}|\,,$$

con lo que no habremos ganado gran cosa.

elementos. Obsérvese que en cuanto $r \geq n$, esta cantidad vale 2^n, todas las listas posibles (por eso la bola de radio 2 en dos dimensiones incluía a las cuatro listas posibles).

3.1.2 Y su Álgebra

Ya conocemos las operaciones de suma y producto módulo 2 definidas en el conjunto $\{0,1\}$ y sus propiedades, ver el ejercicio 2.8 y el ejemplo 2.9, que lo convierten en el cuerpo que hemos dado en llamar $\mathbb{F}_2$. Pasemos ahora al conjunto $\{0,1\}^n$. En él vamos a definir otras dos operaciones, la **suma** y el **producto por escalares** (elementos de $\{0,1\}$). La definición es completamente natural, y se reduce a hacer las operaciones sobre cada coordenada del vector.

- Si $\mathbf{x}$ e $\mathbf{y}$ son dos elementos de $\{0,1\}^n$, su suma se define como

$$\mathbf{x} + \mathbf{y} = (x_1, \ldots, x_n) + (y_1, \ldots, y_n) = (x_1 + y_1, \ldots, x_n + y_n)\,,$$

  donde las sumas de coordenadas se realizan con las reglas[32] de la suma que hemos introducido en $\{0,1\}$.

- Si $\mathbf{x}$ es un elemento de $\{0,1\}^n$ y a un escalar (un elemento de $\{0,1\}$),

$$a \cdot \mathbf{x} = a\,(x_1, \ldots, x_n) = (a\,x_1, \ldots, a\,x_n)\,,$$

  donde las coordenadas se multiplican con las reglas del producto en $\{0,1\}$ que están contenidas en nuestra tabla de multiplicar[33].

Ejercicio 3.4 Mostrar que las operaciones de suma y producto que acabamos de definir satisfacen las siguientes propiedades:

$\mathbf{x} + \mathbf{y} = \mathbf{y} + \mathbf{x}$	Conmutativa de $+$
$(\mathbf{x} + \mathbf{y}) + \mathbf{z} = \mathbf{x} + (\mathbf{y} + \mathbf{z})$	Asociativa de $+$
$\mathbf{x} + \mathbf{0} = \mathbf{0} + \mathbf{x} = \mathbf{x}$	$\mathbf{0}$ es neutro para $+$
$\forall\ \mathbf{x}\ \exists\ \mathbf{y}$ tal que $\mathbf{x} + \mathbf{y} = \mathbf{y} + \mathbf{x} = \mathbf{0}$	Existencia de opuesto para $+$
$1\mathbf{x} = \mathbf{x}$	1 es neutro para el producto
$a(b\mathbf{x}) = (ab)\mathbf{x}$	Asociativa de los productos
$(a + b)\mathbf{x} = a\mathbf{x} + b\mathbf{x}$	Distributiva
$a(\mathbf{x} + \mathbf{y}) = a\mathbf{x} + a\mathbf{y}$	Distributiva

Las letras $\mathbf{x}$, $\mathbf{y}$ y $\mathbf{z}$ indican n-listas cualesquiera de ceros y unos, en tanto que $\mathbf{0}$ representa la lista formada por n ceros. Los números a y b son escalares cualesquiera (0 o 1, no hay mucho para elegir en el cuerpo $\mathbf{F}_2$).

[32] Esto es, módulo 2: $1 + 1 = 0$.

[33] De nuevo, módulo 2, pero para la multiplicación ni se nota la diferencia con las reglas usuales.

Las propiedades contenidas en nuestro último ejercicio permiten asegurar que con las operaciones que hemos introducido el conjunto $\{0,1\}^n$ adquiere la estructura de **espacio vectorial** sobre el cuerpo $\mathbb{F}_2$. Emplearemos la notación $\mathbb{F}_2^n$ para referirnos a este espacio vectorial. El conjunto de vectores es exactamente $\{0,1\}^n$, pero esta nueva notación nos recuerda que, a partir de la suma y el producto en $\mathbb{F}_2$, están definidas sobre él operaciones de suma y producto por escalares que lo dotan de una estructura será de gran utilidad en las aplicaciones a la codificación de información.

A continuación presentaremos algunos otros conceptos interesantes relativos a la estructura de espacio vectorial. Esencialmente, son las mismas ideas a las que estamos acostumbrados en el contexto de $\mathbb{R}^n$ (pero ahora el cuerpo de escalares no es $\mathbb{R}$, sino $\mathbb{F}_2$). El lector que esté familiarizado con los aspectos básicos de la teoría de espacios vectoriales puede obviar la lectura del resto de esta sección, que ha sido incluido sólo para que la exposición resulte esencialmente autocontenida.

• Una propiedad esencial de un espacio vectorial es que dados m vectores, o elementos del conjunto, $\mathbf{x}_1, \ldots, \mathbf{x}_m$, donde m es un número natural cualquiera, puede definirse una **combinación lineal** de ellos. Es decir, una expresión del tipo

$$\lambda_1 \mathbf{x}_1 + \lambda_2 \mathbf{x}_2 + \cdots + \lambda_m \mathbf{x}_m \,,$$

donde $\lambda_1, \ldots, \lambda_m$, son m escalares (en nuestro caso ceros o unos). Naturalemente, esta combinación lineal es un nuevo vector del espacio[34].

• Llamaremos **subespacio vectorial** de $\mathbb{F}_2^n$ a cualquier subconjunto no vacío con la propiedad de que cualquier combinación lineal de sus elementos (con escalares cualesquiera) pertenece al subconjunto[35]. Recordemos que, por ejemplo en $\mathbb{R}^3$, los planos o rectas que pasaban por el origen son subespacios vectoriales (pero un plano que no pasa por el origen no lo es). De hecho, estos son todos los posibles (junto con todo $\mathbb{R}^3$ y el subespacio trivial formado por el punto del origen). Es fácil comprobar que, en particular, el elemento $\mathbf{0} = (0, 0, \ldots, 0)$ pertenece a cualquier subespacio vectorial de $\mathbb{F}_2^n$.

• Dada una colección $\mathbf{x}_1, \ldots, \mathbf{x}_k$ de elementos de $\mathbb{F}_2^n$, el conjunto de todas sus posibles combinaciones lineales, que escribiremos como

$$V = \mathcal{L}(\{\mathbf{x}_1, \ldots, \mathbf{x}_k\}) \,,$$

es un subespacio vectorial de $\mathbb{F}_2^n$. Es lo que llamaremos el **subespacio generado por** los vectores $\mathbf{x}_1, \ldots, \mathbf{x}_k$. Diremos que estos vectores son un

[34] Obsérvese que en este caso el cuerpo de escalares es $\mathbb{F}_2$, y en este conjunto el opuesto de 1, al que es usual indicar con el símbolo -1 es el propio 1. Por lo tanto, sumar es lo mismo que restar.

[35] Otra manera de expresar esto es decir que el conjunto es *cerrado* respecto a las operaciones de suma de vectores y multiplicación por un escalar.

sistema de generadores del subespacio V, o que la familia $\{\mathbf{x}_1, \ldots, \mathbf{x}_k\}$ **genera** V.

• Un conjunto de vectores $\{\mathbf{x}_1, \ldots, \mathbf{x}_k\}$ se dirá **linealmente independiente** si la única manera de escribir el elemento $\mathbf{0}$ como combinación lineal suya es la que corresponde a tomar todos los coeficientes cero. Esto es, si

$$\lambda_1 \mathbf{x}_1 + \cdots + \lambda_k \mathbf{x}_k = \mathbf{0} \quad \Longrightarrow \quad \lambda_1 = \cdots = \lambda_k = 0\,.$$

Una caracterización equivalente es la siguiente: el conjunto $\{\mathbf{x}_1, \ldots, \mathbf{x}_k\}$ es linealmente independiente si ninguno de los vectores se puede escribir como combinación lineal del resto.

• Diremos que un conjunto de vectores $\{\mathbf{x}_1, \ldots, \mathbf{x}_k\}$ es una **base** del subespacio V si es linealmente independiente y genera el conjunto V (como es un generador cualquier elemento de V se puede escribir como combinación lineal de los vectores $\mathbf{x}_1, \ldots, \mathbf{x}_k$. Además, como consecuencia de la independencia lineal, hay sólo una manera de hacerlo). Es fácil comprobar que, aunque la base de un subespacio no es única, todas ellas tienen el mismo número de elementos; a ese número común (el tamaño de las bases) es a lo que llamaremos **dimensión** del subespacio. La dimensión de un subespacio mide el "tamaño" del subespacio; pero la idea de tamaño a que nos referimos hay que entenderla en el contexto de la estructura lineal: da idea de cuántos vectores linealmente independientes "caben", como mucho, en el subespacio.

En el contexto de los cuerpos finitos (como es el caso de $\mathbb{Z}_2$), la dimensión también da idea del número de elementos de que consta un espacio vectorial: un subespacio vectorial de dimensión k en $\mathbb{F}_2^n$ contiene 2^k elementos. Por ejemplo, en un subespacio de dimensión 2 en $\mathbb{F}_2^3$ hay $2^2 = 4$ elementos. En cuerpos con un número infinito de elementos (por ejemplo, $\mathbb{R}$), la situación es distinta: por ejemplo, en $\mathbb{R}^3$ un subespacio de dimensión 2 (un plano por el origen) tiene un número infinito de elementos. Comprobemos que un subespacio V de dimensión k en $\mathbb{F}_2^n$ consta de 2^k elementos. Cada elemento de V será una combinación lineal de los k elementos de una base suya

$$\{\mathbf{x}_1, \cdots, \mathbf{x}_k\}\,,$$

una expresión del tipo

$$\lambda_1 \mathbf{x}_1 + \cdots + \lambda_k \mathbf{x}_k\,.$$

En principio hay 2^k elecciones de parámetros $\lambda_1, \ldots, \lambda_k$ posibles (recordemos que los λ_j sólo pueden tomas los valores 0 y 1)); si probamos que no hay dos elecciones distintas que generen el mismo elemento, habremos terminado. Pero es que si se cumpliera que

$$\lambda_1 \mathbf{x}_1 + \cdots + \lambda_k \mathbf{x}_k = \widetilde{\lambda}_1 \mathbf{x}_1 + \cdots + \widetilde{\lambda}_k \mathbf{x}_k\,,$$

entonces se debería verificar que

$$(\lambda_1 - \widetilde{\lambda}_1)\,\mathbf{x}_1 + \cdots + (\lambda_k - \widetilde{\lambda}_k)\,\mathbf{x}_k = \mathbf{0}\,.$$

Y como los $\mathbf{x}_j$ son elementos de una base, forman un conjunto linealmente independiente; así que la única posibilidad es que todos los coeficientes de la combinación anterior sean nulos. Esto es, que $\lambda_j = \widetilde{\lambda}_j$ para cada $j = 1, \ldots k$.

• La última cuestión que vamos a revisar trata sobre cómo se puede describir un subespacio vectorial V de dimensión k en $\mathbb{F}_2^n$ (o, en general, en un espacio vectorial de dimensión n). Hay básicamente dos maneras distintas de hacerlo:

- La primera es dar una base suya, un conjunto de vectores $\{\mathbf{x}_1, \ldots, \mathbf{x}_k\}$ linealmente independientes que generen V. Y podemos escribir una matriz G de tamaño $k \times n$ cuyas filas sean los elementos de la base. Cada elemento de $\mathbf{v} \in V$ se obtendrá como

  $$(v_1, \ldots, v_n) = (\lambda_1, \ldots, \lambda_k)\,G\,,$$

  para cada elección de parámetros $(\lambda_1, \ldots, \lambda_k)$.

- La otra manera de describir el subespacio mediante ecuaciones, con una matriz H cuyas filas contengan los coeficientes de estas ecuaciones lineales. Cada ecuación que los elementos de V deban satisfacer representa una restricción que "elimina" una dimensión, un grado de libertad. Por lo tanto, para llegar a k dimensiones a partir de las n originales harán falta $n-k$ condiciones, y es necesario además que no haya condiciones redundantes. La matriz será $(n-k) \times n$, (tiene $n-k$ filas, cada una de las cuales expresa una relación que deben satisfacer las n componentes de los vectores del de $\mathbb{F}_2^n$ para estar en el subespacio V de dimensión k) y sus $n-k$ filas tienen que ser linealmente independientes[36] (esta condición expresa la ausencia de redundancia entre las restricciones impuestas por las distintas filas). Se tendrá que

  $$\mathbf{v} \in V \quad \Longleftrightarrow \quad H\,\mathbf{v}^t = \mathbf{0}^t\,,$$

  donde $\mathbf{0}$ es la lista con $n-k$ ceros.

Y la pregunta es: ¿cómo se pasa de G a H (o al revés)? La herramienta adecuada es el bien conocido método de Gauss. Veámoslo con un ejemplo en $\mathbb{F}_2^5$.

[36] Una manera breve de expresar todo esto es decir que H es una matriz $(n-k) \times n$ de rango $n-k$.

Ejemplo 3.5 Consideremos el subespacio vectorial V de dimensión 3 en $\mathbb{R}^5$, que tiene como base

$$\{(1,0,1,1,1),(0,0,1,1,1),(1,1,0,0,0)\}.$$

La matriz G se forma situando los vectores de la base como filas suyas. Y así, para cada elección de parámetros (λ,μ,ν) tendremos un elemento de V:

$$(x_1,\ldots,x_5)=(\lambda,\mu,\nu)\begin{pmatrix}1&0&1&1&1\\0&0&1&1&1\\1&1&0&0&0\end{pmatrix}.$$

Si transponemos esta identidad matricial, obtenemos lo que a veces se llaman ecuaciones paramétricas del subespacio:

$$\begin{pmatrix}1&0&1\\0&0&1\\1&1&0\\1&1&0\\1&1&0\end{pmatrix}\begin{pmatrix}\lambda\\\mu\\\nu\end{pmatrix}=\begin{pmatrix}x_1\\x_2\\x_3\\x_4\\x_5\end{pmatrix}\Longrightarrow\begin{cases}x_1&=&\lambda+\nu\\x_2&=&\nu\\x_3&=&\lambda+\mu\\x_4&=&\lambda+\mu\\x_5&=&\lambda+\mu\end{cases}$$

Miremos ahora al sistema de ecuaciones (en su forma matricial). La pregunta que debemos hacernos es: ¿qué ha de cumplir un dato $(x_1,\ldots,x_5)$ para que el sistema tenga solución, esto es, para que podamos encontrar (λ,μ,ν) de manera que se satisfaga la identidad? Responder a esta pregunta es lo mismo que decidir cuándo el sistema tiene solución (con λ, μ y ν las incógnitas). El método de Gauss nos permite encontrar una matriz que describa el mismo sistema de ecuaciones y de la que podamos leer directamente las condiciones que estamos buscando. Con las operaciones habituales del método de Gauss (intercambiar filas o combinarlas), llegamos a otra matriz ampliada del sistema,

$$\left(\begin{array}{ccc|c}1&0&1&x_1\\0&0&1&x_2\\1&1&0&x_3\\1&1&0&x_4\\1&1&0&x_5\end{array}\right)\longrightarrow\left(\begin{array}{ccc|c}1&0&0&x_1+x_2\\0&1&0&x_1+x_2+x_3\\0&0&1&x_2\\0&0&0&x_3+x_4\\0&0&0&x_3+x_5\end{array}\right)$$

Las filas de ceros nos informan de las condiciones de compatibilidad del sistema:

$$\begin{cases}x_3+x_4&=&0\\x_3+x_5&=&0\end{cases}$$

Éstas son las ecuaciones del subespacio, que podemos escribir también en forma matricial:

$$\underbrace{\begin{pmatrix}0&0&1&1&0\\0&0&0&1&1\end{pmatrix}}_{H}\begin{pmatrix}x_1\\x_2\\x_3\\x_4\\x_5\end{pmatrix}=\begin{pmatrix}0\\0\end{pmatrix}$$

Para recuperar una base del subespacio vectorial, como tenemos dos ecuaciones, lo que hay que hacer es escribir dos coordenadas en función de las restantes, que actuarán como parámetros. En general, eso supone escribir H en su forma escalonada, con el método de Gauss. En este caso, se puede hacer directamente: por ejemplo, despejamos x_4 y x_5 en función del resto (que tomamos como parámetros):

$$\left\{\begin{array}{rcl} x_4 & = & x_3 \\ x_5 & = & x_3 \end{array}\right. \quad \Longrightarrow \quad \left\{\begin{array}{rcl} x_1 & = & \lambda \\ x_2 & = & \mu \\ x_3 & = & \nu \\ x_4 & = & \nu \\ x_5 & = & \nu \end{array}\right.$$

Escribiendo estas ecuaciones en forma matricial, recuperamos la expresión de G:

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix} = \underbrace{\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix}}_{G^t} \begin{pmatrix} \lambda \\ \mu \\ \nu \end{pmatrix} \quad \Longrightarrow \quad G = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \end{pmatrix}.$$

Y si alguien queda preocupado por haber llegado a una matriz G distinta de la inicial, que pruebe a hacer las siguientes operaciones: sumar la primera y la tercera filas de esta matriz. Así se obtiene la primera fila de la original; la segunda fila de la matriz original es simplemente la tercera de ésta; sumando la primera y segunda filas de la nueva matriz recuperamos la tercera de la matriz de partida. Simplemente ocurre que hemos llegado a una base distinta del mismo subespacio vectorial, lo que corresponde a una matriz G diferente. Sin embargo, ambas describen exactamente el mismo subespacio del espacio vectorial $\mathbb{F}_2^5$. ♣

4 Detección y corrección de errores

4.1 Generalidades y primeros ejemplos

Empecemos viendo algunos ejemplos, ya en el lenguaje de las listas de ceros y unos. Supongamos que deseamos transmitir un mensaje que está formado con palabras de un diccionario $\mathcal{P}$ que, digamos, es todo el conjunto $\{0,1\}^k$. La primera opción podría ser enviar los mensajes tal cual están; es decir, considerar que el código es $\mathcal{C} = \{0,1\}^k$. Si enviamos palabras código de éstas a través del canal no vamos a tener opción de ni siquiera detectar los posibles errores, porque *cualquier* palabra que recibamos estará en nuestro diccionario. Necesitamos ser más cuidadosos.

Ejemplo 4.1 UN CONTROL DE PARIDAD.
Éste es un ejemplo semejante a los que ya aparecieron al tratar los códigos de barras, el ISBN, o la prueba del 9, en el sentido de que vamos a agregar *un* carácter de control a las palabras de nuestro diccionario

$$\mathcal{P} = \{0,1\}^k .$$

Lo que haremos será añadir un dígito más, que informe sobre si el número de unos que contiene la palabra es par o impar; por ejemplo, añadimos un 0 si es par, y un 1 si es impar. Si tenemos una palabra $\mathbf{p} = (x_1, \ldots, x_k)$, una manera de ver si tiene un número par o impar de unos es calcular, módulo 2, el valor de

$$\sum_{j=1}^{k} x_j .$$

Este valor será 0 si hay un número par de unos, y 1 si hay un número impar. Así que codificaremos de la siguiente manera:

$$\begin{array}{rcl} f \,:\, \mathcal{P} = \{0,1\}^k & \to & \mathcal{C} \subset \{0,1\}^{k+1} \\ \mathbf{p} = (x_1, \ldots, x_k) & \mapsto & \mathbf{c} = (x_1, \ldots, x_k, x_{k+1}) \,, \end{array}$$

con

$$x_{k+1} = \sum_{j=1}^{k} x_j .$$

Por supuesto, esta suma debe entenderse en el sentido de la aritmética módulo 2. Por ejemplo, las palabras

$$(0,1,1,0), \quad (1,0,0,0)$$

de $\{0,1\}^4$ deberían codificarse como

$$(0,1,1,0,\mathbf{0}), \quad (1,0,0,0,\mathbf{1})$$

respectivamente. Hemos representado en negrita el caracter de control que agrega la codificación.

Las palabras código son ahora listas de $k+1$ posiciones, elementos de $\{0,1\}^{k+1}$. Pero no son todas: sólo estarán aquéllas que cumplan la condición

$$\sum_{j=1}^{k+1} x_j \equiv 0 \pmod 2,$$

que son exactamente la mitad de las 2^{k+1} listas de $\{0,1\}^{k+1}$. En este proceso estamos añadiendo información suplementaria, lo que se traduce en que las palabras del código ya no son todas las posibles.

Ejercicio 4.2 Codificar con este sistema todas las palabras de $\{0,1\}^3$. Hallar el conjunto de palabras de $\{0,1\}^4$ que forman el código $\mathcal{C}$ resultante. ♣

Ejemplo 4.3 UN CÓDIGO DE REPETICIÓN.
Tomemos ahora las $2^4 = 16$ palabras de

$$\mathcal{P} = \{0,1\}^4 .$$

Codificaremos cada una de esas palabras enviándolas dos veces seguidas:

$$\begin{array}{rcl} f \,:\, \mathcal{P} = \{0,1\}^4 & \to & \mathcal{C} \subset \{0,1\}^8 \\ (x_1\, x_2\, x_3\, x_4) & \mapsto & (x_1\, x_2\, x_3\, x_4\, x_1\, x_2\, x_3\, x_4)\,. \end{array}$$

Observemos que este código duplica la cantidad de información a transmitir (cada palabra original, de cuatro posiciones, se transforma en una de ocho). Además, y esto es importante, nuestro diccionario no contiene todas las palabras de ocho posiciones, sólo aquéllas que cumplen la condición de repetición (pregunta: ¿cuántas contiene? Más importante todavía: ¿qué proporción de las palabras de $\{0,1\}^8$ están en el código?). Desde luego, es un código que permite detectar errores: basta comparar las dos "mitades" de la lista que llega. Si recibimos

$$\left\| \begin{array}{cccc|cccc} 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 \end{array} \right\|$$

estaremos seguros de que se ha producido un error. Pero, ¿cuál de las posiciones ha cambiado? Parece claro que el error está en la primera posición, pero, ¿de cuál de las mitades? Es un código que no nos permite corregir. Por cierto, recibir una palabra de nuestro código no garantiza que no se hayan producido errores: podría haber cambiado, por ejemplo, la cuarta posición en las dos mitades (aunque entonces sería necesario que se cometieran al menos *dos* errores). ♣

Ejemplo 4.4 UN CÓDIGO DE REPETICIÓN (MEJOR).
Continuemos trabajando con palabras de $\mathcal{P} = \{0,1\}^4$; pero ahora las codificamos enviando cada una de ellas tres veces seguidas. La codificación sería

$$\begin{array}{rcl} f : \mathcal{P} = \{0,1\}^4 & \longrightarrow & \mathcal{C} \subset \{0,1\}^{12} \\ (x_1\,x_2\,x_3\,x_4) & \longmapsto & (x_1\,x_2\,x_3\,x_4\,x_1\,x_2\,x_3\,x_4\,x_1\,x_2\,x_3\,x_4)\,. \end{array}$$

De nuevo, nuestro código no contiene todas las palabras de $\{0,1\}^{12}$. La cantidad de palabras del código es la misma que en el ejemplo anterior ($2^4 = 16$), pero ahora representan una fracción menor del total. Podremos detectar errores comparando los "tercios" de cada palabra transmitida. Por ejemplo, si recibimos

0	1	1	0	0	1	1	0	0	1	1	1

podemos deducir que se ha producido un error, porque esta palabra no pertenece al código; algo sencillo de comprobar porque el tercer tercio —formado por los últimos cuatro caracteres— es distinto de los dos primeros —las cuaternas de caracteres entre los lugares primero y cuarto, y quinto y octavo—. Diríamos además que el error está en la última posición, la duodécima. Así que estaríamos tentados de interpretar que la palabra originalmente transmitida era $(0,1,1,0)$. Si supiéramos que hay sólo un error podríamos asegurar que nuestra conjetura es correcta, y, en esta situación de un sólo error, ya lo estaríamos corrigiendo. Lo que nos ha permitido esta mejora (pasar de detectar a corregir el error) es que *hemos separado* las palabras del diccionario (eso sí, a cambio de irnos a palabras código más largas). Quizás este concepto de "separación" todavía no sea evidente; veamos otros ejemplos más ilustrativos. ♣

Ejemplo 4.5 UN CÓDIGO CON PALABRAS SEPARADAS
Supongamos ahora que nuestro código sólo consta de dos palabras,

0	1	1	1

0	0	1	0

que difieren en dos posiciones. Si recibimos una palabra que no esté en el código, por ejemplo (0110), detectaremos el error. Pero no vamos a poder saber cuál fue: podríamos haber emitido (0111) o (0010), y en ambos casos un único error nos habría llevado a la palabra transmitida. Observemos que para que al emitir (0111) recibamos (0010) han de cometerse 2 errores: para confundir dos palabras del código se requieren 2 errores. Este código detecta un error (pero en general no más de uno, con dos errores podríamos estar recibiendo una palabra del código que no fuera la realmente emitida) y no permite corregir ninguno. ♣

Ejemplo 4.6 UN CÓDIGO CON PALABRAS AÚN MÁS SEPARADAS
Ahora tenemos un código con dos palabras

0	0	0	0

1	1	1	1

Para confundir dos palabras del código (es decir, emitir una del código y recibir otra distinta, también del código) han de cometerse cuatro errores. Así que este código detecta hasta tres errores. Y, además, nos va a permitir corregir: supongamos que recibimos la lista (0111). Es claro que se han producido errores, porque no es una palabra de nuestro diccionario. Y cualquiera apostaría a que en realidad estábamos transmitiendo (1111) y a que se ha producido un error. La otra opción sería haber emitido (0000) y tener tres errores; esto parece mucho menos probable. La misma situación tendríamos si recibiéramos la lista (0010), que interpretaríamos como (0000). Ahora bien, si recibiéramos (1100), aún sabiendo que ha habido errores, no podríamos decidir si originalmente era (0000) o (1111). Es decir, este código, en el que para confundir dos palabras se necesitan cuatro errores, detecta hasta tres errores, y parece capaz de corregir uno. ♣

Antes de seguir adelante, vale la pena enfatizar lo que significa la expresión **corregir hasta m errores** en la última frase del ejemplo anterior. Con el mismo sentido la emplearemos en el resto del texto: cualquier palabra que haya sido transmitida con no más de m errores es interpretada correctamente. Con las palabras que tienen más de m errores puede ocurrir cualquier cosa. Por ejemplo, podremos aceptarlas sin modificaciones si coinciden con otra palabra del código; o incluso agregar más errores para interpretarlas como una palabra del código diferente de la que se transmitió.

Podemos extraer una serie de conclusiones de estos ejemplos: interesa que nuestro diccionario no contenga a todas las palabras posibles; de hecho, es conveniente que sea lo más "disperso" posible. Si las palabras del código están muy alejadas entre sí, vamos a poder definir algún criterio para corregir los posibles errores. Podemos separar las palabras del código introduciendo información extra, lo que se traduce, en general, en que las palabras código son de mayor longitud que las palabras originales de nuestro mensaje. Por ejemplo, si nuestro mensaje tiene palabras que son listas de k ceros y unos podemos pensar que las palabras código serán listas de longitud n (elementos de $\{0,1\}^n$), con $n > k$. El que el código no contenga todas las palabras posibles se reflejará en que, efectivamente, $\mathcal{C}$ sea un subconjunto de $\{0,1\}^n$ (y no todo el conjunto).

Parece claro entonces que un concepto importante en un código $\mathcal{C}$ es la llamada **distancia mínima** entre palabras del código, que denotaremos con $d_{\mathcal{C}}$, y que se define como la menor distancia, en el sentido de la distancia de Hamming, que puede haber entre palabras diferentes del código.

O, equivalentemente, la distancia entre las dos palabras del código más cercanas entre sí (por supuesto, también aquí estamos considerando palabras diferentes). La fórmula que define esta distancia mínima es

$$d_{\mathcal{C}} = \min_{\substack{\mathbf{x},\mathbf{y} \in \mathcal{C} \\ \mathbf{x} \neq \mathbf{y}}} d(\mathbf{x},\mathbf{y})\,.$$

Si recordamos cómo hemos definido la distancia de Hamming, resulta que $d_{\mathcal{C}}$ es el número mínimo de posiciones en que pueden diferir dos palabras cualesquiera de nuestro código.

En términos de esta distancia mínima podemos ya establecer un resultado que mide la capacidad de detección de errores de un código:

Teorema 1 *Un código $\mathcal{C}$ cuya distancia mínima sea $d_{\mathcal{C}}$ es capaz de* ***detectar*** *hasta $d_{\mathcal{C}} - 1$ errores en la transmisión de una palabra.*

DEMOSTRACIÓN. Sea $\mathbf{c}$ una palabra del código. Si introducimos hasta $d_{\mathcal{C}} - 1$ errores en esta palabra no vamos a obtener otra palabra del código, ya que la palabra del código más próxima a $\mathbf{c}$ está a una distancia de $\mathbf{c}$ que es mayor o igual que $d_{\mathcal{C}}$. Por lo tanto, la nueva palabra no está en nuestro diccionario (el código $\mathcal{C}$) y detectaremos que se han producido errores. □
Si se producen $d_{\mathcal{C}}$ o más errores puede que ya no seamos capaces de detectar el error.

Por el contrario, si el número de errores cometidos es menor que $d_{\mathcal{C}}/2$ no sólo podremos detectarlos: será posible *corregirlos e identificar correctamente la palabra del código que se deseó transmitir.* Para verlo llamemos r al número de errores cometidos. Tenemos entonces $r < d_{\mathcal{C}}/2$. Entonces la distancia entre la palabra $\mathbf{x}$ recibida y la palabra $\mathbf{c}$ emitida es justamente r, y $\mathbf{x}$ está en la bola $B_r(\mathbf{c})$ que tiene centro $\mathbf{c}$ y radio r. Como la distancia mínima del código es $d_{\mathcal{C}} > 2r$ cualquier bola $B_r(\mathbf{c}')$ es disjunta con $B_r(\mathbf{c})$, tal como se muestra en la figura 1. Por lo tanto, la distancia de $\mathbf{x}$ a cualquier

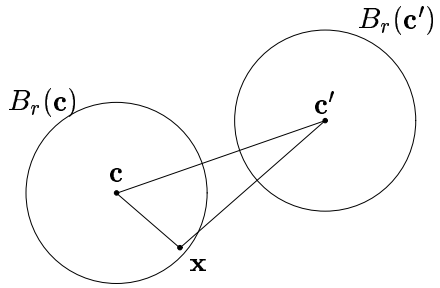


Figura 1: Corrección de errores buscando el "vecino más próximo".

otra palabra $\mathbf{c}'$ debe ser mayor que r, y la palabra enviada $\mathbf{c}$ es la que está a menor distancia de $\mathbf{x}$ entre todas las del código.

Esta observación es la base de un procedimiento de descodificación que pasamos a describir a continuación. Supongamos que ya hemos establecido un procedimiento de codificación

$$f : \mathcal{P} = \{0,1\}^k \quad \longrightarrow \quad \mathcal{C} \subset \{0,1\}^n ,$$

tal que cada palabra de $\mathcal{C}$ proviene de una única palabra de $\mathcal{P}$ (con esto estamos diciendo que $f : \mathcal{P} \to \mathcal{C}$ es biyectiva, ver la discusión de la página 23). El proceso de **descodificación** consiste en recuperar la información transmitida a partir de lo que se ha recibido (que no necesariamente coincidirá con lo enviado, lo que hace que toda esta historia tenga cierto interés) y tiene esencialmente dos partes:

1. si hemos recibido una palabra que está en el código $\mathcal{C}$ entonces basta aplicar la función f^{-1}, inversa de la función f que hace la codificación, para conocer de qué palabra proviene. Recordemos que

   $$f^{-1} : \mathcal{C} \to \{0,1\}^k$$

   está definida porque f es biyectiva;

2. para descodificar las palabras que *no* estén en el código $\mathcal{C}$ utilizaremos el criterio del **vecino más próximo** que vamos a describir en términos de la distancia en el código $\mathcal{C}$. Supongamos que recibimos una cierta palabra $\mathbf{x}$. Entonces le asignaremos la palabra $\mathbf{c}$ del código que cumpla que

   $$d(\mathbf{x},\mathbf{c}) < d(\mathbf{x},\mathbf{c}') \quad \text{para toda } \mathbf{c}' \in \mathcal{C} \text{ tal que } c' \neq c. \tag{1}$$

   Quizás nos hayamos pasado de optimistas al hablar de "la palabra", porque esa tal palabra del código bien podría no ser única. Este problema debe ser tenido en cuenta a la hora del diseño del código, y veremos que hay códigos que tienen la propiedad de que (1) caracteriza $\mathbf{c}$ sin ambigüedad para cualquier $\mathbf{x}$. Cuando esto no ocurre el criterio de descodificación es incompleto. Lo usual es "saltarse" una palabra a la que no pueda asignarse una palabra del código, o bien pedir que se reemita, si es posible. Por supuesto, si la palabra ha sido transmitida correctamente, este criterio hace que le asignemos la palabra del código adecuada[37]. Una vez identificada la palabra de $\mathcal{C}$ que resultó del proceso de codificación, ya podemos aplicar f^{-1} para conocer la palabra correspondiente de $\mathcal{P}$.

Ahora podemos retomar las ideas que se esquematizan en la figura 1 y establecer un resultado que mide la capacidad de corrección de un código.

[37] además de estar justificado por la intuición, esta manera de descodificar buscando el "vecino mas cercano" puede fundamentarse en base a consideraciones estadísticas. Al respecto, ver el ejercicio 5.7, página 83

En su enunciado utilizaremos la siguiente notación: si x es un número real cualquiera designaremos con $\lfloor x \rfloor$ su parte entera. Es decir, el mayor entero que es menor o igual que x. Por ejemplo, $3 = \lfloor \pi \rfloor$.

Teorema 2 *Un código $\mathcal{C}$ cuya distancia mínima sea $d_{\mathcal{C}}$ es capaz de* ***corregir***[38] *(con el criterio del vecino más próximo) hasta*

$$\left\lfloor \frac{d_{\mathcal{C}} - 1}{2} \right\rfloor$$

errores en la transmisión de una palabra.

DEMOSTRACIÓN. Supongamos que emitimos una palabra $\mathbf{c}$ del código y que recibimos $\mathbf{x}$; y que se han producido exactamente r errores en la transmisión, con

$$r \leq \left\lfloor \frac{d_{\mathcal{C}} - 1}{2} \right\rfloor. \tag{2}$$

Entonces hay exactamente r posiciones en que la palabra recibida $\mathbf{x}$ difiere de la palabra $\mathbf{c}$, por lo que, teniendo en cuenta la definición de la distancia de Hamming, tenemos $d(\mathbf{x}, \mathbf{c}) = r$. Mostraremos que no hay ninguna otra palabra del código a distancia menor o igual que r de $\mathbf{x}$, lo que implica $\mathbf{c}$ es la única palabra de $\mathcal{C}$ que está a distancia r de $\mathbf{x}$. Todas las demás están más lejos. En consecuencia, el criterio del vecino más próximo nos permitirá recuperar la palabra original $\mathbf{c}$.

La desigualdad triangular permite estimar por debajo la distancia de x a cualquier palabra $\mathbf{c}'$ del código, ya que de

$$d(\mathbf{c}', \mathbf{x}) + d(\mathbf{c}, \mathbf{x}) \geq d(\mathbf{c}', \mathbf{c})$$

podemos despejar[39]

$$d(\mathbf{c}', \mathbf{x}) \geq d(\mathbf{c}', \mathbf{c}) - d(\mathbf{c}, \mathbf{x}). \tag{3}$$

Pero tenemos algo de información sobre los términos del miembro de la derecha. La distancia entre las palabras $\mathbf{c}'$ y $\mathbf{c}$ del código es mayor o igual que la distancia mínima $d_{\mathcal{C}}$. Todavía podemos dar una cota inferior de $d_{\mathcal{C}}$ en términos del número de errores r, usando (2). Tenemos

$$2r \leq 2 \left\lfloor \frac{d_{\mathcal{C}} - 1}{2} \right\rfloor \leq 2 \, \frac{d_{\mathcal{C}} - 1}{2} = d_{\mathcal{C}} - 1 \; < d_{\mathcal{C}}.$$

Concluimos entonces que

$$d(\mathbf{c}', \mathbf{c}) \geq d_{\mathcal{C}} > 2r.$$

[38] Ver los comentarios inmediatamente después del ejemplo 4.6, en la página 38, acerca del significado del término "corregir" en este enunciado.

[39] Esta nueva desigualdad corresponde a la siguiente afirmación: "la diferencia entre dos lados de un triángulo es más pequeña que el tercer lado".

Por otra parte, sabemos que $d(\mathbf{c}, \mathbf{x}) = r$. Usando esta información en (3) obtenemos

$$d(\mathbf{c}', \mathbf{x}) > 2r - r = r,$$

lo que completa la prueba. Antes de darla por terminada conviene mencionar que la información que hemos manejado es exactamente la misma que está contenida en la figura 1, pero le hemos dado un tratamiento analítico basado en la desigualdad triangular que satisface la distancia de Hamming. □

Ejercicio 4.7 Calcular la distancia mínima para los códigos basados en un control de paridad (ver el ejemplo 4.1), y para el código de repetición que codifica palabras de longitud k repitiéndolas m veces. Determinar la cantidad de errores que estos códigos pueden detectar y corregir.

Los códigos correctores no sólo permiten detectar errores y eliminarlos. También son capaces de recuperar información perdida, o *borrones* en la transmisión. Por ejemplo, si pretendemos enviar la palabra (0101), pero el destinatario recibe (01 1) diremos que se ha producido un borrón en la tercera posición. El siguiente ejercicio muestra la capacidad de corregir borrones de un código. Más adelante discutiremos los procedimientos para determinar cuáles eran los dígitos perdidos.

Ejercicio 4.8 CORRECCIÓN DE BORRONES

1. Mostrar que un código con distancia mínima d_c es capaz de recuperar correctamente, usando el criterio del vecino más próximo, una palabra que tiene hasta $d_c - 1$ borrones.
2. Generalizar el resultado de la primera parte, mostrando que el código es capaz de corregir una palabra con b borrones y r errores, siempre que $b + 2r < d_c$.

4.2 Parámetros de los códigos y cota de Hamming

Es usual escribir el parámetro $d_{\mathcal{C}}$, un parámetro relevante para el código $\mathcal{C}$, en términos de un nuevo parámetro e que es esencialmente igual al número máximo de errores que el código es capaz de corregir. Si $d_{\mathcal{C}}$ es impar lo escribimos en la forma $d_{\mathcal{C}} = 2e + 1$, entonces el código detecta hasta $2e$ errores y corrige, con el criterio de vecino más próximo, hasta e errores. Escribimos $d_{\mathcal{C}} = 2e$ cuando es par. Entonces corrige hasta $e - 1$ errores, como se comprueba sin dificultad. Este comentario muestra que los códigos con distancia mínima impar son más eficientes para la corrección de errores, dado que códigos con distancias mínimas $2e+1$ y $2e+2$ tienen exactamente la misma capacidad de corrección. Por esta razón, la mayoría de los códigos que consideraremos tendrán distancia mínima impar.

Un parámetro que permite comparar la separación entre las palabras de un código con la longitud de las palabras es la **distancia mínima relativa**,

$$\delta_{\mathcal{C}} = \frac{d_{\mathcal{C}}}{n}.$$

El valor de la distancia mínima en un código $\mathcal{C} \subset \{0,1\}^n$ siempre es menor o igual que n, por lo que $0 < \delta_{\mathcal{C}} \leq 1$. Para una longitud de palabras dada, nos interesará encontrar códigos con distancia mínima lo más grande posible; esto es, con $\delta_{\mathcal{C}}$ cercano a 1.

Otras variables de interés para la teoría que estamos desarrollando tienen que ver con cuánta información adicional se introduce a la hora de codificar. Llamaremos **tasa de transmisión** del código $\mathcal{C}$ a la razón $R_{\mathcal{C}}$ entre la longitud de las palabras antes y después de ser codificadas[40]. Si estamos codificando palabras de longitud k con palabras código de longitud n tendremos

$$R_{\mathcal{C}} = \frac{k}{n}\,.$$

La **redundancia** del código será

$$1 - R_{\mathcal{C}} = \frac{n-k}{k},$$

que mide cuál es la fracción del total de la información que hemos agregado con el propósito de corregir errores. Parece razonable suponer que cuanto menor sea la tasa de transmisión (y mayor, por tanto, la redundancia), más errores seremos capaces de detectar y/o corregir. A cambio, la longitud de las palabras código será mayor.

En resumen, podemos estudiar un código $\mathcal{C}$ utilizando los dos parámetros $R_{\mathcal{C}}$ y $\delta_{\mathcal{C}}$, que son números entre 0 y 1. Y parece claro que un buen código será aquél que tenga ambos parámetros grandes: $R_{\mathcal{C}}$ próximo a 1 quiere decir que no aumentamos mucho el tamaño del mensaje al codificarlo, mientras que un $\delta_{\mathcal{C}}$ grande nos permitirá corregir más errores. Pero es fácil darse cuenta, como se intuye en los ejemplos que hemos descrito, que estos dos objetivos van en sentidos contrarios: $R_{\mathcal{C}}$ grande implica que la codificación no añade mucha información extra, que es justo la que permite corregir errores (porque separa las palabras).

Parece claro que esta capacidad de corrección, dada por lo "separadas" que están las palabras del código, limita el número de palabras que este código puede tener: si el diccionario incluye muchas de las 2^n palabras posibles, éstas no podrán estar muy separadas. El siguiente ejercicio contiene un resultado que da una medida cuantitativa de esta relación entre "separación de palabras" y "número de palabras del código".

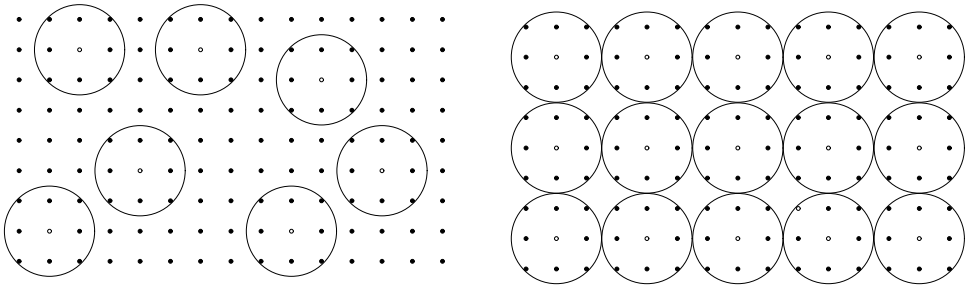
Ejercicio 4.9 Cota de Hamming y códigos perfectos
Considerermos un código $\mathcal{C}$ formado por n-listas de ceros y unos cuya distancia mínima sea $d_{\mathcal{C}} \geq 2e+1$.

[40] El uso de la letra R para esta cantidad proviene del término inglés "rate".

1. Entonces
$$|\mathcal{C}| \sum_{j=0}^{e} \binom{n}{j} \leq 2^n \,. \tag{4}$$
Sugerencia: calcular la cantidad de palabras en una bola de radio e, y usar el hecho de que las bolas de radio e centradas en palabras del código son disjuntas dos a dos.

2. Diremos que un código es *perfecto* si satisface la igualdad en (4). Mostrar que un código es perfecto si y sólo si $\{0,1\}^n$ es unión de las bolas de radio e centradas en palabras del código. Mostrar que un código perfecto descodifica de manera única cualquier palabra, y que si el número r de errores cometido en una palabra satisface $r > e$ entonces la descodificación es errónea.

Los códigos perfectos son óptimos en el siguiente sentido: supongamos que queremos codificar pálabras de longitud k , y que existe un código perfecto con palabras de longitud n y distancia mínima $2e+1$ que nos permite hacerlo. Entonces, cualquier otro código con la misma distancia mínima que no sea perfecto debe utilizar palabras de longitud mayor que n, y, en consecuencia, su redundancia será mayor y la transmisión más costosa. Dejamos como ejercicio para el lector interesado la verificación de esta propiedad de los códigos perfectos. Estos códigos, de indudable interés, son poco frecuentes (veremos algunos ejemplos más adelante). Para hacernos una idea geométrica de qué significa un código perfecto representemos[41] las listas de $\{0,1\}^n$ como puntos del plano; y en torno a cada palabra del código, dibujamos la bola de radio e:



Tenemos dos códigos (formados por las palabras que son centro de las bolas dibujadas) con distancia mínima $2e+1$. Ambos cumplen que las bolas de radio e son disjuntas dos a dos, así que corregirán hasta e errores. Pero el de la derecha, además, es perfecto, pues todos los puntos de $\{0,1\}^n$ están en una bola de radio e centrada en algún punto del código.

[41] Los díbujos que aparecen a continuación no son demasiado fieles a la verdadera geometría de los códigos y los espacios $\mathbb{F}_2^n$. Estos espacios tienen en realidad el aspecto de los vértices de un cubo en un espacio de dimensión n. En cualquier caso, un dibujo es *siempre* una representación del objeto dibujado, y su utilidad depende más de nuestra capacidad de interpretarlo, que de la fidelidad que tenga. Una representación demasiado fiel puede incluso ser inútil, como el mapa del imperio en el cuento de Borges.

Resumimos a continuación algunas de las características que buscaremos a la hora de diseñar códigos:

- la longitud de las palabras código, n, debería ser pequeña, para que sea "barato" transmitir. Al menos no debería ser mucho mayor que la longitud k de las palabras originales: el parámetro $R_{\mathcal{C}}$ debería ser lo más próximo a 1 posible.

- Querremos que $d_{\mathcal{C}}$ sea grande ($\delta_{\mathcal{C}}$ cercano a 1), para poder corregir muchos errores. Esto es, las palabras del código deberían estar lo más separadas posible.

- Las palabras del diccionario no deberían ser todas las posibles n-listas de ceros y unos. Pero $|\mathcal{C}|$ debería ser lo más grande posible: cuantas más palabras tenga, más útil será el código.

Como ya hemos dicho varias veces, estos objetivos son contrapuestos: si hay poca redundancia las palabras estarán poco separadas. O si las palabras están muy separadas, entonces no podrá haber muchas (la cota de Hamming nos da una medida de esta relación). Una estrategia habitual es fijar dos de los parámetros, n y $d_{\mathcal{C}}$, y tratar de maximizar el tercero, $|\mathcal{C}|$ (los códigos perfectos son ejemplos óptimos para este problema).

Pero además hay otras cuestiones que deberíamos tener en cuenta, y que justifican en parte los códigos lineales que veremos más adelante:

- en principio, describir un código $\mathcal{C}$ en $\{0,1\}^n$ requiere dar las $|\mathcal{C}|$ palabras que lo forman. Y calcular $d_{\mathcal{C}}$ exige calcular las $\binom{|\mathcal{C}|}{2}$ distancias entre las palabras (y luego calcular el mínimo de ellas). A veces podremos hacer todo esto de una manera más eficaz.

- Y los procesos de codificación y descodificación deberían ser sencillos.

Todas estas operaciones se simplifican cuando el código tiene una cierta estructura. Es muy difícil calcular todas las distancia entre palabras del código, o definir la función de codificación por algún procedimiento sistemático e implementable, si el código es una "nube de puntos" colocados más o menos al azar en el conjunto $\{0,1\}^n$. Podremos sistematizar todas estas operaciones recurriendo a la *estructura* que tiene este conjunto. Ya vimos como definir en él operaciones que lo convierten en un espacio vectorial que hemos dado en llamar $\mathbb{F}_2^n$, y como medir distancias en este espacio. A continuación desarrollaremos códigos que utilizan fuertemente la estructura lineal de $\mathbb{F}_2^n$: los códigos lineales.

4.3 Códigos lineales

En esta sección vamos a explotar la estructura lineal de $\{0,1\}^k$ y $\{0,1\}^n$ para construir códigos y procesos de codificación y descodificación. A partir

de ahora emplearemos la notación $\mathbb{F}_2^k$ y $\mathbb{F}_2^n$, porque deseamos enfatizar que haremos uso de que estos conjuntos son espacios vectoriales sobre el cuerpo $\mathbb{F}_2$ una vez que se introducen las operaciones de suma de vectores y producto por un escalar que hemos presentado en la sección 3.1. Esto nos permitirá emplear cierta intuición geométrica que proviene de la aplicación de la estructura vectorial de $\mathbb{R}^2$ y $\mathbb{R}^3$ al estudio de la geometría del plano y el espacio respectivamente, y construir funciones de codificación eficientes que responden a sencillas fórmulas algebraicas. En resumen, la estructura lineal proporcionará las herramientas para manipular grandes cantidades de información con relativamente poco esfuerzo. Por ejemplo, recordemos que el espacio $\mathbb{F}_2^n$ consta de 2^n elementos y tiene dimensión[42] n. Su tamaño como conjunto es 2^n, en el sentido de que contiene 2^n elementos; pero como espacio vectorial su "tamaño" es n, que es un número mucho más pequeño que 2^n. Por ejemplo, 500 no es un número demasiado grande, pero 2^{500} es un número mayor que un 3 seguido de 150 ceros[43]. En esta sencilla observación esta basada mucha de la economía que obtendremos al introducir los códigos lineales.

Pasemos ahora a definir los códigos lineales $\mathcal{C}$ en $\mathbb{F}_2^n$.

Definición 2 *Un* ***código lineal*** $\mathcal{C}$ *es un subespacio vectorial de* $\mathbb{F}_2^n$.

Si la dimensión del subespacio que constituye un código $\mathcal{C}$ es k diremos que $\mathcal{C}$ es un $[n,k]$-código lineal. Como todo código, tendrá una distancia mínima, digamos d. Entonces hablaremos de un $[n,k,d]$-código lineal.

A riesgo de ser redundantes, digámoslo una vez más: la ventaja de un código lineal es que tiene estructura. No es simplemente una larga lista de palabras. Con una base (que es relativamente poca información, k palabras) tendremos descritas todas las 2^k palabras del código. Además, podremos construir procesos de codificación y descodificación basados en la estructura lineal, que serán fáciles de manejar porque habrá una descripción sencilla de la función de codificación

$$f : \mathcal{P} = \mathbb{F}_2^k \to \mathcal{C} \subset \mathbb{F}_2^n \, .$$

[42] La dimensión es igual al número de vectores de una base. Estos espacios tienen una base *canónica*

$$B = \{\mathbf{e}_1, \ldots, \mathbf{e}_n\} \, ,$$

donde $\mathbf{e}_j$ es la lista de n posiciones que tiene un uno en la posición j y ceros en el resto. Recordemos además que la lista $\mathbf{0} = (0, \ldots, 0)$ es, por supuesto, el elemento neutro del espacio.

[43] Este número resulta mayor que el número de partículas que, según las estimaciones actuales, contiene todo el universo. En cambio 500 partículas no abultan mucho.

Las funciones exponenciales con una base mayor que 1, como $k \mapsto 2^k$, $x \mapsto e^x$, etcétera, tienen un crecimiento vertiginoso en cuanto la variable que aparece en el exponente aumenta por encima de 1.

Ejemplo 4.10 CONTROLES DE PARIDAD
La introducción de dígitos de control de paridad da lugar a un código lineal. En efecto, la condición que satisfacen las palabras $(x_1, x_2, \ldots, x_k, x_{k+1})$ que están en el código es

$$x_1 + x_2 + \cdots + x_k + x_{k+1} = 0,$$

Esta ecuación lineal (que debe entenderse en el sentido de las operaciones módulo 2) define un subespacio de dimensión k en $\mathbb{F}_2^{k+1}$. Como se trata de una sola ecuación sólo se elimina *un* grado de libertad, y la dimensión del código es exactamente una unidad menor que la del espacio lineal que lo contiene. ♣

Ejemplo 4.11 CÓDIGOS DE REPETICIÓN.
También los códigos de repetición son lineales. Por ejemplo, si para codificar palabras de longitud 4 las repetimos 3 veces, las palabras de nuestro código quedan caracterizadas por las ecuaciones lineales

$$x_1 = x_5 = x_9, \quad x_2 = x_6 = x_{10}, \quad x_3 = x_7 = x_{11}, \quad x_4 = x_8 = x_{12},$$

que expresan en forma algebraica la condición de repetición. Naturalmente, una base del código está formada por los vectores

$$\begin{gathered}(1,0,0,0,1,0,0,0,1,0,0,0),\\ (0,1,0,0,0,1,0,0,0,1,0,0),\\ (0,0,1,0,0,0,1,0,0,0,1,0),\\ (0,0,0,1,0,0,0,1,0,0,0,1).\end{gathered}$$

Este código tiene parámetros $[12, 4, 3]$ y su redundancia es $2/3$. Observemos que, en general, la tasa de transmisión de un $[n, k]$ código lineal es k/n, y su redundancia es $(n-k)/n$.

Ejercicio 4.12 Mostrar que para este código hay una desigualdad estricta en la cota de Hamming, por lo que no es perfecto.

Otro sencillo código de repetición se obtiene de la siguiente manera: para enviar mensajes formados con las "palabras" 0 y 1 las codificamos repitiendo cada símbolo n veces. Nuestro código se reduce entonces a

$$\{(0,\ldots,0),(1,\ldots,1)\},$$

que forma un subespacio lineal de dimensión uno en $\mathbb{F}_2^n$. La distancia mínima (y única) en este código es n. Será, pues, un $[n, 1, n]$-código lineal, con muchísima redundancia. Si $n = 2e + 1$, será capaz de corregir hasta e errores.

Ejercicio 4.13 Mostrar que para cada n impar, el código de repetición de parámetros $[n, 1, n]$ es un código perfecto (estos códigos son los llamados códigos perfectos triviales). ♣

4.3.1 Cálculo de la distancia mínima

Abordaremos ahora el problema de calcular la distancia mínima en el contexto de los códigos lineales. El procedimiento más directo para determinar esta distancia mínima consiste en aplicar directamente su definición: para todos las parejas formadas por dos elementos distintos cualesquiera del código, es decir, listas $\mathbf{x}$ e $\mathbf{y}$ de ceros y unos, calcularemos su distancia. Luego buscaremos el valor mínimo de todas estas distancias.

Recordemos que la distancia entre dos listas

$$\mathbf{x} = (x_1, \ldots, x_n), \quad \mathbf{y} = (y_1, \ldots, y_n),$$

es igual al número de posiciones en que ambas listas no coinciden. También es igual al número de posiciones con entrada distinta de 0 en la diferencia $\mathbf{x}-\mathbf{y}$ entre. Esta observación es más o menos obvia, y se justifica completamente observando que

$$\mathbf{x} - \mathbf{y} = (x_1 - y_1, \ldots, x_n - y_n)\,.$$

Para cada j entre 1 y n tenemos

$$\begin{array}{ll} x_j - y_j = 0 & \text{si y sólo si} \quad x_j = y_j, \\ x_j - y_j \neq 0 & \text{si y sólo si} \quad x_j \neq y_j, \end{array}$$

lo que demuestra nuestra afirmación. En conclusión, el cálculo de la distancia entre dos listas puede hacerse manipulando sólo una lista: la diferencia entre ambas, y contando cuántas de sus entradas difieren de 0. Como utilizaremos esta cantidad repetidas veces resultará útil introducir un nombre para designarla, así que llamaremos **peso** de una lista $\mathbf{x}$ al número de entradas no nulas que contiene. Indicaremos este número con la notación[44] $\omega(\mathbf{x})$. Por ejemplo

$$\omega((0,1,1,0,1)) = 3,$$

y si indicamos con $\mathbf{e}_j$, $j = 1, 2, \ldots, n$, a los vectores de la base canónica de cualquier espacio $\mathbb{F}_2^n$ tenemos

$$\omega(\mathbf{e}_j) = 1, \quad j = 1, 2, \ldots, n.$$

Teniendo en cuenta las definiciones de la distancia de Hamming y el peso de un vector, resulta

$$d(\mathbf{x}, \mathbf{y}) = \omega(\mathbf{x} - \mathbf{y}),$$

para cualquier par $(\mathbf{x}, \mathbf{y})$ de listas de la misma longitud.

[44] Para los amantes de los excesos de notación escribamos, si $\mathbf{x} = (x_1, x_2 \ldots, x_n)$,

$$\omega(\mathbf{x}) = |\{x_j \neq 0 \,:\, 1 \leq j \leq n\}|\,.$$

Ejercicio 4.14 Mostrar que $d(\mathbf{x}, \mathbf{0}) = \omega(\mathbf{x})$.

Veamos por qué resulta adecuado introducir este concepto y cuál es su utilidad en el marco de los códigos lineales. Supongamos que queremos calcular la distancia mínima de un cierto código $\mathcal{C}$ en $\mathbb{F}_2^n$. En principio deberíamos calcular las distancias entre todos los pares de palabras del código, o, lo que es equivalente, los pesos de todas las diferencias, y buscar el más pequeño de todos estos pesos. Podemos escribir este cálculo en la forma

$$d_{\mathcal{C}} = \min_{\substack{\mathbf{x} \neq \mathbf{y} \\ \mathbf{x}, \mathbf{y} \in \mathcal{C}}} \omega(\mathbf{x} - \mathbf{y}) \,. \tag{5}$$

Deberíamos formar entonces $\binom{|\mathcal{C}|}{2}$ parejas distintas con elementos del código (el orden es irrelevante para este cálculo, porque la distancia es simétrica) y luego calcular las restas y los pesos correspondientes. Pero si el código es lineal, entonces *todas las diferencias entre listas del código están en el código*, y todas las palabras del código se pueden escribir como diferencia de dos palabras (esto es obvio, porque la lista $\mathbf{0}$ está en cualquier código lineal). Por lo tanto, el conjunto de las diferencias $\mathbf{x} - \mathbf{y}$ no tiene que ser calculado, porque es el propio código $\mathcal{C}$. La palabra $\mathbf{0}$ queda fuera del conjunto de diferencias, porque sólo nos interesan las que provienen de palabras distintas, de modo que sólo tenemos que calcular los pesos de las $|\mathcal{C}| - 1$ palabras no nulas del código. La fórmula que exprésa el cálculo de la distancia mínima para un código lineal $\mathcal{C}$ se reduce entonces a

$$d_{\mathcal{C}} = \min_{\mathbf{x} \in \mathcal{C} \setminus \{\mathbf{0}\}} \omega(\mathbf{x}) \,.$$

La simplificación respecto a (5) es doble:

- no tenemos que calcular diferencias entre palabras del código $\mathcal{C}$, porque nos bastará examinar las palabras no nulas de $\mathcal{C}$.
- el número de pesos a calcular se reduce muchísimo. Hemos visto que pasa de

  $$\binom{|\mathcal{C}|}{2} = \frac{|\mathcal{C}|(|\mathcal{C}| - 1)}{2}$$

  a $|\mathcal{C}| - 1$. Si el código tiene 8 palabras estos números son 28 y 7, y para un código de 16 palabras tendríamos 120 y 15. En estos ejemplos la diferencia ya es notoria, y se hace más importante en códigos con muchas palabras[45].

Ejercicio 4.15 Calcular las distancias mínimas de los controles de paridad y los códigos de repetición.

[45] La razón es que $\binom{|\mathcal{C}|}{2}$ crece como un cuadrado de $|\mathcal{C}|$, en tanto que $|\mathcal{C}| - 1$ lo hace linealmente.

Todavía veremos otra manera de calcular eficazmente esta distancia mínima en un código lineal, cuando introduzcamos las matrices de control en la sección 4.3.4.

4.3.2 Codificación con códigos lineales

La estructura lineal del código tambien puede ser explotada a la hora de codificar y descodificar. En efecto, veremos a continuación que escogiendo una función lineal de codificación

$$\begin{array}{rccl} f\ :\ \mathcal{P} = \mathbb{F}_2^k & \to & \mathcal{C} \subset \mathbb{F}_2^n \\ \mathbf{p} & \mapsto & f(\mathbf{p}) = \mathbf{c}\,, \end{array}$$

toda la información relativa a la codificación quedará almacenada en una matriz de k filas y n columnas[46], que permitirá calcular muy fácilmente la función f para cada una de las 2^k palabras de $\mathcal{P}$.

Una codificación lineal f puede definirse a través de una base cualquiera del código, tal como se explica a continuación. Dado un $[n,k]$-código lineal $\mathcal{C}$, podremos escoger una base en él, digamos $\{\mathbf{x}_1, \ldots, \mathbf{x}_k\}$. Y codificar una palabra

$$\mathbf{p} = (p_1, p_2, \cdots, p_k)$$

cualquiera usando sus entradas como coeficientes de una combinación lineal de la base, de la siguiente manera:

$$\mathbf{p} \mapsto f(\mathbf{p}) = p_1\mathbf{x}_1 + p_2\mathbf{x}_2 + \cdots + p_k\mathbf{x}_k.$$

Más agradable aún es el hecho de que f admite una expresión matricial. Si llamamos x_{ij} al símbolo que lleva en la posición j el i-ésimo vector de la base, podemos construir la matriz

$$G = \begin{pmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_k \end{pmatrix} = \begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{k1} & x_{k2} & \cdots & x_{kn} \end{pmatrix},$$

[46] Podría hacerse exactamente lo mismo con una matriz de n filas y k columnas. Esto depende de la convención que se escoja para representar las transformaciones lineales por medio de una matriz. Vale la pena mencionar que en estas notas usaremos la notación corriente en la teoría de códigos lineales, y multiplicaremos vectores fila a izquierda de una matriz para calcular sus imágenes por una transformación lineal. Esto es lo contrario de lo que suele hacerse en los cursos de álgebra lineal, en los que la notación habitual implica que hay que multiplicar un vector columna a la derecha de las matrices que representan funciones lineales. Todo esto es un poco enojoso, pero para nada difícil. Para pasar de una notación a otra sólo hay que trasponer todas las matrices. En estas notas tomaremos las traspuestas sin dar mayores explicaciones cada vez que nos resulte conveniente. Por ejemplo, para aplicar el método de Gauss de resolución de sistemas lineales.

a las que nos referiremos como una **matriz generatriz** del código. Cualquier combinación lineal de las filas de esta matriz puede obtenerse multiplicándola a la izquierda por el vector fila formado por los coeficientes de la combinación[47]. En otras palabras, el vector

$$\mathbf{c} = f(p) = p_1\mathbf{x}_1 + p_2\mathbf{x}_2 + \cdots + p_k\mathbf{x}_k$$

también puede escribirse en la forma

$$\mathbf{c} = (p_1, p_2, \ldots, p_k) \begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{k1} & x_{k2} & \cdots & x_{kn} \end{pmatrix}.$$

La matriz G tiene dimensiones $k \times n$. Hay k filas formadas por las coordenadas de los vectores longitud n de una base del código $\mathcal{C}$. Al multiplicarla a la izquierda por $\mathbf{p}$ obtenemos una combinación lineal de sus filas, que es un vector

$$\mathbf{c} = (c_1, c_2, \ldots, c_n)$$

que está en el código, y al que hemos escogido como el resultado de codificar $\mathbf{p}$. En resumen, *toda la información* relativa a la función de codificación f que hemos fabricado *está en la matriz* G.

Ejercicio 4.16 Mostrar que la codificación que acabamos de describir define una función $f : \mathcal{P} \to \mathcal{C}$ que es biyectiva.

Recíprocamente, podemos representar de esta manera cualquier función lineal

$$f : \mathcal{P} \to \mathcal{C}$$

por medio de una matriz cuyas filas sean los vectores

$$f(\mathbf{e}_1), f(\mathbf{e}_2), \ldots, f(\mathbf{e}_k),$$

donde cada vector $\mathbf{e}_i$, para $i = 1, 2 \ldots, k$, tiene un 1 en el lugar i y 0 en las restantes $k-1$ posiciones[48]. Si la función f es biyectiva entonces las filas de esta matriz forman una base de $\mathcal{C}$.

Ejercicio 4.17 Verificar todas las afirmaciones del último párrafo.

[47] En la página 62 mostramos un resultado similar para las columnas de una matriz: el resultado de multiplicar una matriz a la derecha por un vector columna es el mismo que hacer una combinación lineal de las columnas de la matriz empleando como coeficientes las entradas del vector.

[48] Esta familia de vectores es la *base canónica* de $\mathbb{F}_2^k$.

Ejemplo 4.18 CODIFICACIÓN EN UN CÓDIGO DE REPETICIÓN
Ya hemos construido, en el ejemplo 4.11 una base para el código que consiste en repetir 3 veces las palabras de cuatro dígitos. A partir de esa base podemos construir la matriz generatriz

$$G = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

Por ejemplo, si calculamos el producto $(1,0,1,1)G$ obtenemos como resultado $(1,0,1,1,1,0,1,1,1,0,1,1)$, y vemos que la matriz produce un vector de longitud 12 con tres "copias" del original. ♣

Ejemplo 4.19 CODIFICACIÓN PARA LOS CONTROLES DE PARIDAD
Un control de paridad agrega a un k-vector de ceros y unos un 1 o un 0 en el lugar $k+1$, según que haya una cantidad impar o par de unos. Hay una sencilla fórmula para escribir esto en la aritmética módulo 2:

$$x_{k+1} = x_1 + x_2 + \cdots + x_k.$$

Por lo tanto, la función f que codifica es

$$(x_1, x_2, \ldots, x_k) \mapsto (x_1, x_2, \ldots, x_k, x_1 + x_2 + \cdots + x_k).$$

Podemos generar la matriz de la codificación calculando f sobre los vectores de la base canónica de $\mathbb{F}_2^k$, y obtenemos

$$\begin{pmatrix} 1 & 0 & 0 & \cdots & 0 & 1 \\ 0 & 1 & 0 & \cdots & 0 & 1 \\ 0 & 0 & 1 & \cdots & 0 & 1 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & 1 \end{pmatrix}.$$

Ejemplo 4.20 UN CÓDIGO LINEAL $[7,4,3]$
Para construir[49] este código empezaremos empleando un lenguaje distinto. Consideremos el conjunto $\mathcal{A} = \{1, \ldots, 7\}$ y formemos la siguiente familia de

[49]Quizás esta no sea la forma más breve de caracterizar este código. De hecho, en el ejemplo 4.26 mostramos que puede describirse como el resultado de agregar tres caracteres de control a una palabra de cuatro dígitos. Sin embargo, llama la atención sobre el aspecto combinatorio de la teoría de códigos, que ha quedado bastante soslayado en estas páginas. Fíjense en la familia de tríos que hemos formado; son la solución al siguiente problema combinatorio: tenemos siete personas, y cada día de la semana salen a pasear tres de ellas. Queremos que cada persona disfrute de la compañía de todas las demás (¡hemos de fomentar la conversación!), así que ha de compartir paseo con todas ellas. Pero sólo una vez con cada una, tampoco es cuestión de repetir. Y claro, parece justo que todos se den el mismo número de paseos, tres en este caso. Puede parecer un poco rebuscado, pero en

3-subconjuntos suyos:

$$\mathcal{F}_1 = \{1,2,4\} \quad \mathcal{F}_2 = \{2,3,5\} \quad \mathcal{F}_3 = \{3,4,6\} \quad \mathcal{F}_4 = \{4,5,7\}$$
$$\mathcal{F}_5 = \{5,6,1\} \quad \mathcal{F}_6 = \{6,7,2\} \quad \mathcal{F}_7 = \{7,1,3\}$$

Es fácil comprobar que cualquier subconjunto de dos elementos de $\mathcal{A}$ está en un único miembro de esta familia. Y que cada elemento de $\mathcal{A}$ está en exactamente tres miembros de esta familia. Traduzcamos cada $\mathcal{F}_i$ a listas de ceros y unos con la receta obvia: colocaremos un 1 en la posición j si el elemento j forma parte de $\mathcal{F}_i$:

$$\begin{array}{rcl}
\mathcal{F}_1 = \{1,2,4\} & \longrightarrow & \mathbf{a}_1 = \boxed{1\ 1\ 0\ 1\ 0\ 0\ 0} \\
\mathcal{F}_2 = \{2,3,5\} & \longrightarrow & \mathbf{a}_2 = \boxed{0\ 1\ 1\ 0\ 1\ 0\ 0} \\
\mathcal{F}_3 = \{3,4,6\} & \longrightarrow & \mathbf{a}_3 = \boxed{0\ 0\ 1\ 1\ 0\ 1\ 0} \\
\mathcal{F}_4 = \{4,5,7\} & \longrightarrow & \mathbf{a}_4 = \boxed{0\ 0\ 0\ 1\ 1\ 0\ 1} \\
\mathcal{F}_5 = \{5,6,1\} & \longrightarrow & \mathbf{a}_5 = \boxed{1\ 0\ 0\ 0\ 1\ 1\ 0} \\
\mathcal{F}_6 = \{6,7,2\} & \longrightarrow & \mathbf{a}_6 = \boxed{0\ 1\ 0\ 0\ 0\ 1\ 1} \\
\mathcal{F}_7 = \{7,1,3\} & \longrightarrow & \mathbf{a}_7 = \boxed{1\ 0\ 1\ 0\ 0\ 0\ 1}
\end{array}$$

Tenemos pues siete elementos de $\mathbb{F}_2^7$, pero no son linealmente independientes: es fácil comprobar que

$$\mathbf{a}_5 = \mathbf{a}_1 + \mathbf{a}_2 + \mathbf{a}_3, \quad \mathbf{a}_6 = \mathbf{a}_2 + \mathbf{a}_3 + \mathbf{a}_4 \quad \text{y} \quad \mathbf{a}_7 = \mathbf{a}_1 + \mathbf{a}_2 + \mathbf{a}_4,$$

y que el conjunto $\{\mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3, \mathbf{a}_4\}$ es linealmente independiente.

Formamos entonces el siguiente $[7,4]$-código lineal: es el subespacio vectorial de dimensión 4 en $\mathbb{F}_2^7$ que tiene $\{\mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3, \mathbf{a}_4\}$ como una base; su matriz generatriz será

$$G = \begin{pmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{pmatrix} \tag{6}$$

Y toda palabra del código se podrá escribir como

$$(a_1, a_2, a_3, a_4) \begin{pmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{pmatrix}$$

realidad este tipo de problemas tienen aplicaciones importantes, por ejemplo en el diseño de experimentos. Digamos que una revista especializada en automóviles quiere probar, en días sucesivos, cinco modelos diferentes, y dispone de cinco personas para hacerlo. ¿Cómo organizarlo para que cada persona pruebe una vez cada vehículo (y así tener todas las opiniones), teniendo en cuenta que cada día una persona sólo puede probar un automóvil y que cada vehículo sólo puede ser probado una vez al día? De estas construcciones se ocupa una rama de la Combinatoria, la Teoría de Diseños combinatorios.

donde (a_1, a_2, a_3, a_4) es cualquier vector de parámetros en $\mathbb{F}_2^4$.

¿Cuál es la distancia mínima en este código? Tendremos que calcular el peso de todas las $2^4 - 1 = 15$ palabras no nulas del código. Los elementos de la base tienen todos peso 3 (hay 4 de ellos). Hay $\binom{4}{2}$ que se obtienen como combinación lineal (suma) de dos de la base. En principio, estos elementos podrían tener peso hasta 6, pero todos los pares tienen un 1 en la misma posición, así que el peso de todos ellos es 4. Los cuatro elementos que se obtienen sumando tríos de vectores de la base tienen todos peso 3. Y la suma de los cuatro vectores produce un vector de peso 4. Así que la distancia mínima es 3. Corregirá, por tanto, un error; y es fácil comprobar que es perfecto:

$$|\mathcal{C}| \left(\binom{7}{0} + \binom{7}{1} \right) = 2^4 \, (1 + 7) = 2^4 \, 2^3 = 2^7 \, .$$

El código anterior consta de $2^4 = 16$ palabras. Para calcular su distancia mínima no hemos tenido que calcular las $\binom{16}{2} = 120$ distancias entre ellas; nos ha bastado con evaluar los 15 pesos de las palabras no nulas. En la sección 4.3.4 mostraremos un procedimiento alternativo para hacer el cálculo.

El código que estamos describiendo es perfecto y tiene distancia mínima 3. Por lo tanto, cualquier palabra de siete dígitos tiene una palabra del código a distancia menor o igual que 1, que es la que interpretaríamos al descodificar una transmisión por el criterio del vecino más próximo. De eso trata nuestro próximo ejercicio.

Ejercicio 4.21 BUSCANDO EL VECINO MÁS PRÓXIMO
Hallar la palabra del código más cercana[50] a cada una de las siguientes:

$$(1,1,1,1,1,1,1), \quad (1,1,1,1,0,1,1), \quad (1,0,1,1,0,1,1), \quad (1,1,1,0,1,0,1).$$

Ejercicio 4.22

1. Dar una matriz que permita codificar palabras de dos dígitos con el código lineal generado por $\{(1,1,0,1,0), (1,0,1,0,1)\}$. Hallar todas las palabras del código y su distancia mínima.

2. En un mensaje codificado de esta manera se recibe $(0,1,1,1,1)$, ¿cuál es la palabra de dos dígitos que se transmitió?

3. Si recibimos $(1,0,1,1,1)$, ¿qué debemos interpretar? ¿Y si es $(0,0,0,1,1)$ la lista recibida?

[50] Luego presentaremos algunas técnicas para hacer esto en forma sistemática, pero ahora se trata de resolver el problema con las herramientas que cada cual pueda desarrollar.

4.3.3 Descodificación: cuando no hay errores

Revisemos en qué consistía un proceso de codificación: tenemos un diccionario de palabras $\mathcal{P}$, que son k-listas de ceros y unos. Y las transformamos en palabras de un código $\mathcal{C} \subset \mathbb{F}_2^n$ mediante una cierta receta f, la codificación:

$$\begin{aligned} f \,:\, \mathcal{P} \subset \mathbb{F}_2^k &\to \mathcal{C} \subset \mathbb{F}_2^n, \\ \mathbf{p} &\mapsto f(\mathbf{p}) = \mathbf{c}. \end{aligned}$$

Esta palabra $\mathbf{c}$ es la que se transmite, y quien la reciba deseará, en general, determinar cuál fue la palabra $\mathbf{p}$ que ha sido codificada. Si la palabra $\mathbf{c}$ fue transmitida correctamente[51], sin errores ni borrones, sólo tenemos que invertir el proceso de codificación para calcular $\mathbf{p} = f^{-1}(\mathbf{c})$.

Para un código $\mathcal{C}$ general tendremos que conocer la tabla que define la asignación f; por ejemplo, si tenemos que $\mathcal{P} = \{0,1\}^k$, tendremos que conocer las imágenes por f de las 2^k palabras del diccionario. Pero si el código es lineal $[n,k]$, no hará falta almacenar tanta información: bastará la matriz G generatriz del código que corresponde a la codificación con la que estamos trabajando. Sabemos que dada una palabra de $\mathbb{F}_2^k$, digamos $\mathbf{p} = (p_1, \ldots, p_k)$, la palabra $\mathbf{c} = (c_1, \ldots, c_n)$ del código que le corresponde viene dada, como ya hemos visto, por

$$(c_1, \ldots, c_n) = (p_1, \ldots, p_k)\, G\,. \tag{7}$$

Si conocemos $\mathbf{c}$ y deseamos calcular $\mathbf{p}$ todo lo que tenemos que hacer es "despejar" el vector $\mathbf{p}$ de (7), lo que se reduce a la resolución de un sistema lineal de ecuaciones. Vamos a ver esto en un ejemplo.

Ejemplo 4.23 Retomemos el código $[7,4,3]$ del ejemplo 4.20, para el que habíamos calculado una matriz generatriz G. Dada una palabra $\mathbf{c}$ del código, trataremos de determinar de qué palabra $\mathbf{p}$ proviene. Hagámoslo en general, para cualquier palabra $\mathbf{c} = (c_1, c_2, \ldots, c_7)$, cosa que no aumenta en absoluto la dificultad de los cálculos. Sabemos que tenemos que resolver un sistema lineal, que escribimos en la forma

$$\left(\begin{array}{cccc|c} 1 & 0 & 0 & 0 & c_1 \\ 1 & 1 & 0 & 0 & c_2 \\ 0 & 1 & 1 & 0 & c_3 \\ 1 & 0 & 1 & 1 & c_4 \\ 0 & 1 & 0 & 0 & c_5 \\ 0 & 0 & 1 & 0 & c_6 \\ 0 & 0 & 0 & 1 & c_7 \end{array}\right),$$

[51] En caso contrario, a veces tendremos que interpretar palabras que *no* están en el códigos, a las que no podremos, en consecuencia, aplicar f^{-1}. Para estas palabras tendremos que buscar primero la palabra del código "más próxima", y luego aplicar a esta nueva palabra el proceso de descodificación que discutimos en esta sección. El tratamiento de las palabras con errores se hará en la sección 4.3.6

que se presta a resolverlo con el método de eliminación gaussiana. Por cierto, esta expresión de la ecuación es esencialmente la que se obtiene trasponiendo (7), y cada columna corresponde a una de las incógnitas p_i, $i = 1, 2, 3, 4$. Luego de aplicar cuidadosamente el procedimiento de eliminación obtenemos (ahorramos los pasos intermedios que el lector puede verificar. De paso recordemos que la aritmética que estamos usando es módulo 2):

$$\left.\begin{array}{cccc|l} 1 & 0 & 0 & 0 & c_1 \\ 0 & 1 & 0 & 0 & c_1 + c_2 \\ 0 & 0 & 1 & 0 & c_1 + c_2 + c_3 \\ 0 & 0 & 0 & 1 & c_2 + c_3 + c_4 \\ 0 & 0 & 0 & 0 & c_1 + c_3 + c_4 + c_5 \\ 0 & 0 & 0 & 0 & c_1 + c_2 + c_3 + c_6 \\ 0 & 0 & 0 & 0 & c_2 + c_3 + c_4 + c_7 \end{array}\right. . \tag{8}$$

Esta expresión nos da bastante información. Lo primero que nos dice es que

$$\begin{aligned} p_1 &= c_1, \\ p_2 &= c_1 + c_2, \\ p_3 &= c_1 + c_2 + c_3, \\ p_4 &= c_2 + c_3 + c_4. \end{aligned}$$

De modo que cada palabra $\mathbf{c} = (c_1, c_2, \ldots, c_7)$ del código debe descodificarse como

$$\mathbf{p} = f^{-1}(\mathbf{c}) = (c_1, c_1 + c_2, c_1 + c_2 + c_3, c_2 + c_3 + c_4).$$

Pero, ¿las componentes c_5, c_6 y c_7 del vector $\mathbf{c}$ no intervienen para nada en esto? En realidad sí lo hacen. Estos números aparecen en las tres últimas ecuaciones de (8), que son

$$\begin{aligned} c_1 + c_3 + c_4 + c_5 &= 0, \\ c_1 + c_2 + c_3 + c_6 &= 0, \\ c_2 + c_3 + c_4 + c_7 &= 0, \end{aligned}$$

y que constituyen una *condición de compatibilidad* para que el sistema tenga solución. Dado que sólo podemos descodificar palabras $\mathbf{c}$ que estén en el código estas ecuaciones constituyen en realidad una condición de pertenencia al código para las palabras de $\mathbb{F}_2^7$. Volveremos sobre este aspecto de la teoría de códigos lineales al discutir las *matrices de control* en la sección 4.3.4, por lo que no extendemos más estos comentarios.

Ejercicio 4.24 Recibimos $(1, 1, 1, 1, 1, 1, 1)$, ¿Cuál es la palabra que se codificó? ¿Y si recibimos $(1, 1, 1, 0, 1, 1, 1)$, o $(1, 1, 1, 0, 1, 0, 1)$? ♣

El procedimiento de descodificación que hemos descrito no es demasiado complicado, pero puede resultar laborioso. Es una buena noticia saber que

puede simplificarse enormemente escogiendo de manera adecuada la matriz G. Supongamos que la matriz generatriz G del código es de la forma

$$G = \left(\boxed{I_{k\times k}} \; \boxed{A_{k\times(n-k)}} \right) , \tag{9}$$

donde $I_{k\times k}$ representa a la matriz identidad de dimensiones $k \times k$. En este caso, la codificación da lugar a

$$(p_1, \ldots, p_k)\, G = (p_1, \ldots, p_k, c_{k+1}, \ldots, c_n)\,.$$

Es decir, los primeros k dígitos de la palabra codificada corresponden a la palabra original y el resto de las posiciones recogen la redundancia que añade la codificación. Parece claro que esto será útil a la hora de descodificar: basta mirar las primeras k posiciones para conocer la palabra codificada. Cuando una matriz generatriz G tenga este aspecto, diremos que está en **forma estándar**, y el proceso de codificación correspondiente se llamará **sistemático**. Subrayemos el hecho de que una codificación sistemática se limita a agregar $n - k$ dígitos de control a la palabra original, dejando las primeras k posiciones sin ninguna modificación.

Ejemplo 4.25 Las matrices de codificación que introdujimos en los ejemplos del código de repetición $[12, 4, 3]$ y del control de paridad ya están en forma estándar. ♣

Ya habíamos observado que dar una matriz de codificación es equivalente a dar una base del código. Nuestro problema es construir una base del código tal que su matriz de codificación esté en forma estándar. Afortunadamente, esto puede hacerse aplicando el procedimiento de eliminación gaussiana sobre las filas de una matriz de codificación G cualquiera. Subrayemos que la eliminación permite pasar de una matriz $n \times k$ con un bloque $k \times k$ cualquiera en sus primera k columnas, a una matriz en forma estándar (con una excepción, que discutiremos en el ejemplo 4.29). Además, y es esto lo que lo vuelve útil a nuestros propósitos, si partimos de una matriz cuyas filas son una base del código $\mathcal{C}$ obtendremos luego de aplicar cada paso de la eliminación gaussiana una matriz cuyas filas también son una base de $\mathcal{C}$. En consecuencia, la matriz con la que el proceso termina también tiene esta propiedad. Verifiquemos esto para los pasos elementales de la eliminación, que son:

- cambiar el orden de las filas. Esto se traduce en cambiar el orden de los vectores de la base. Obviamente, da lugar a una nueva base para $\mathcal{C}$.

- Sumar a una fila $\mathbf{v}$ otra fila $\mathbf{w}$. Con esto estaríamos cambiando el par de filas $\mathbf{v}$, $\mathbf{w}$, por el par $\mathbf{v}+\mathbf{w}$, $\mathbf{w}$. Dado que

$$\mathbf{v} = (\mathbf{v}+\mathbf{w}) + \mathbf{w}$$

(recordemos que estamos trabajando con una aritmética módulo 2)[52] podemos recuperar el par original a partir del nuevo, lo que asegura que seguimos teniendo una base del espacio.

Veamos ahora como opera esto en un ejemplo concreto.

Ejemplo 4.26 Mostraremos ahora como la matriz G de los ejemplos 4.20-4.23 puede llevarse a una forma estándar aplicando el proceso de eliminación que acabamos de describir. Recordemos que

$$G = \begin{pmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{pmatrix}.$$

Empezamos bastante bien, porque no hay unos por debajo de la diagonal. Comenzamos a eliminar hacia arriba: a la primera fila le sumamos la segunda y obtenemos

$$\begin{pmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{pmatrix}.$$

En el paso siguiente sumamos la tercera fila a la primera, y la tercera más la cuarta a la segunda. La matriz queda

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{pmatrix}.$$

Para terminar, a la tercera fila sumamos la cuarta, y obtenemos la matriz

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{pmatrix},$$

que permite codificar palabras de cuatro dígitos en forma sistemática.

[52] En general, para aplicar la eliminación gaussiana hace falta multiplicar algunas filas por escalares no nulos. En nuestra aritmética módulo 2 el único escalar no nulo es el 1, así que podemos obviar esta operación.

Ejercicio 4.27 Hacer otra vez el ejercicio 4.24, pero usando esta nueva codificación.

Vale la pena observar que la nueva matriz generatriz que hemos calculado hace que codifiquemos de la siguiente manera:

$$(p_1, p_2, p_3, p_4) \mapsto (p_1, p_2, p_3, p_4, p_1 + p_3 + p_4, p_1 + p_2 + p_3, p_2 + p_3 + p_4),$$

que puede verse como el resultado de agregar los tres dígitos de control

$$p_1 + p_3 + p_4, \quad p_1 + p_2 + p_3, \quad p_2 + p_3 + p_4,$$

a la palabra original. Tenemos así una descripción alternativa a la que propusimos originalmente para este código $[7, 4, 3]$. ♣

Ejercicio 4.28 Hallar una matriz en forma estándar para el código del ejercicio 4.22

Ejemplo 4.29 En este ejemplo mostraremos que las operaciones de eliminación gaussiana que describimos antes pueden no bastar para obtener una matriz de codificación sistemática. Codificaremos palabras de $\mathbb{F}_2^2$ con palabras del código de $\mathbb{F}_2^4$ que queda definido por la matriz

$$G = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix}.$$

En este caso no se puede obtener una codificación sistemática haciendo operaciones entre filas: necesitamos una operación extra, la de intercambiar columnas. Es evidente que si intercambiamos la segunda columna con la tercera obtenemos una matriz G' que está en forma estándar. Pero esta matriz corresponde a un código diferente. Esto es algo sencillo de comprobar: el código que corresponde a la matriz G es

$$\{(0,0,0,0), (1,1,0,0), (0,0,1,1), (1,1,1,1)\},$$

en tanto que el de G' es

$$\{(0,0,0,0), (1,0,1,0), (0,1,0,1), (1,1,1,1)\}.$$

Pero las propiedades del nuevo código son las mismas que las del código original. En realidad ambos son códigos de repetición, la única diferencia entre ambos es la forma en que escribimos la repetición, y sus parámetros son $[4, 2, 2]$. ♣

Con las operaciones del método de Gauss sólo producimos un cambio de base en el subespacio lineal generado por las columnas de la matriz generatriz, así que el código es exactamente el mismo (aunque la codificación, la

receta f, cambiará). Pero, tal como vimos, intercambiar columnas cambia el código con el que estamos trabajando.

Sin embargo, pese a que una operación de intercambio de columnas cambia el código, las propiedades que tiene el nuevo código van a ser las mismas. Supongamos que tenemos un $[n, k]$-código lineal definido por su matriz generatriz G, y formamos una nueva matriz G' intercambiando las columnas i y j de la matriz original. Esta nueva matriz define un código lineal de la misma dimensión, k, y el efecto sobre las palabras código es el siguiente: si una palabra $(p_1, \ldots, p_k)$ se codifica como $(\ldots, c_i, \ldots, c_j, \ldots)$ con la codificación original, entonces será $(\ldots, c_j, \ldots, c_i, \ldots)$ con la segunda. Esto es, intercambia los símbolos de las dos posiciones correspondientes en todas las palabras del código. Esto quiere decir que, pese a que los código sean distintos, comparten los parámetros característicos de los códigos: n, k y $d_{\mathcal{C}}$ (obsérvese que la operación no cambia el peso de las palabras).

Con las operaciones del método de Gauss, más la del intercambio de columnas está garantizado que, dado un código de matriz generatriz G, podemos pasar a uno equivalente (en el sentido de que tiene los mismos parámetros característicos), que tiene una matriz generatriz G de la forma buscada:

$$G = \left(\boxed{k \times k} \; \boxed{k \times (n-k)} \right) \longrightarrow G' = \left(\boxed{I_{k \times k}} \; \boxed{A_{k \times (n-k)}} \right)$$

Ejercicio 4.30 Completar la prueba de que dado un código lineal siempre puede obtenerse un código que tiene los mismos parámetros y que admite una codificación sistemática. Sugerencia: una matriz generatriz de un código $[k, n]$ debe tener k columnas linealmente independientes.

En la sección 6.2, página 87, mostramos una interesante aplicación de las matrices generatrices y los procedimientos de codificación sistemáticos: el intercalado de códigos.

4.3.4 Matrices de control

Un código lineal $\mathcal{C}$ con parámetros $[n, k]$ es un subespacio de dimensión k del espacio vectorial $\mathbb{F}_2^n$. Los códigos útiles satisfacen $k < n$, de modo que el código está estrictamente contenido en el conjunto de las palabras de longitud n, y hay palabras que no están en el código. Como el código es lineal la condición de pertenencia puede escribirse en término de ecuaciones lineales, que admiten una expresión matricial. A la matriz de estas ecuaciones

se le llama **matriz de control del código**. Veamos primero un ejemplo, retomando el código $[7, 4, 3]$ con que hemos estado trabajando.

Ejemplo 4.31 Al calcular la inversa de la codificación para el código $[7, 4, 3]$ en el ejemplo 4.23 llegamos a expresar el sistema lineal en la forma (8), cuyas tres últimas ecuaciones son

$$\begin{array}{rcl} c_1 + c_3 + c_4 + c_5 & = & 0, \\ c_1 + c_2 + c_3 + c_6 & = & 0, \\ c_2 + c_3 + c_4 + c_7 & = & 0. \end{array} \tag{10}$$

Estas ecuaciones expresan la condición para que el sistema se pueda resolver, lo que es completamente equivalente a decir que la palabra

$$\mathbf{c} = (c_1, c_2, c_3, c_4, c_5, c_6, c_7)$$

provenga de alguna palabra

$$\mathbf{p} = (p_1, p_2, p_3, p_4)$$

a través del procedimiento de codificación. Pero éstas son justamente las palabras que están en nuestro código $\mathcal{C}$. Por lo tanto, una palabra de $\mathbb{F}_2^7$ está en $\mathcal{C}$ si y sólo si satisface (10). Estas ecuaciones lineales pueden escribirse en forma matricial, como

$$H\,\mathbf{c}^t = \mathbf{0}^t,$$

donde

$$H = \begin{pmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{pmatrix}$$

La matriz H es la matriz de control para este código. ♣

En general, cualquier código lineal $[k, n]$, es decir, cualquier subespacio de dimensión k de un espacio de dimensión n, puede describirse por medio de $n-k$ ecuaciones lineales independientes, que admiten una escritura matricial en términos de una matriz H de dimensión $(n-k) \times n$, cuyas filas son linealmente independientes. Esta matriz es una **matriz de control** del código. Notemos entonces que las matrices de control tienen la propiedad de que

$$\mathbf{c} = (c_1, c_2 \ldots, c_k) \in \mathcal{C} \Longleftrightarrow H\,\mathbf{c}^t = \mathbf{0}^t\,.$$

Ejemplo 4.32 MATRIZ DE CONTROL PARA LOS CONTROLES DE PARIDAD Ya habíamos observado que en un código que incorpora un control de paridad un vector $(x_1, \cdots, x_k)$ se codifica como $(x_1, \cdots, x_k, x_{k+1})$, de forma tal que

$$x_1 + \cdots + x_k + x_{k+1} = 0.$$

Está claro entonces que

$$H = (1\ 1 \ldots\ 1\ 1),$$

con dimensión $1 \times (k+1)$, es la matriz de control de este código. ♣

Ejercicio 4.33 Para el código de repetición $[12, 4, 3]$, y el código $[5, 2, 3]$ del ejercicio 4.22, hallar matrices de control.

Además de proveer una descripción bastante simple de las palabras del código, las matrices de control tienen aplicaciones a la hora de descodificar mensajes que pueden contener errores y/o borrones, y para el cálculo de la distancia mínima del código. A continuación mostramos cómo puede calcularse la distancia mínima de un código a partir de la matriz de control. En la sección 4.3.6 resultará clara su utilidad para el proceso de descodificación.

Comencemos por destacar el siguiente hecho evidente: al escoger una palabra cualquiera $\mathbf{c}$ y hacer el producto $H\mathbf{c}^{\mathrm{t}}$ se obtiene un vector que es una combinación lineal de las columnas de H, como se muestra a continuación:

$$\left(\boxed{\mathbf{h}_1}\ \boxed{\mathbf{h}_2}\ \cdots\ \boxed{\mathbf{h}_n} \right) \begin{pmatrix} c_1 \\ \vdots \\ c_n \end{pmatrix} = c_1\,\mathbf{h}_1 + \cdots + c_n\mathbf{h}_n.$$

Hemos llamado $\mathbf{h}_1, \ldots, \mathbf{h}_n$ a las columnas de H, y los coeficientes de la combinación son las entradas del vector $\mathbf{c}$. Si la palabra $\mathbf{c}$ está en el código el resultado de $H\mathbf{c}^{\mathrm{t}}$ es el vector nulo. Salvo en el caso trivial, y completamente inútil, en que el código solo contenga la palabra formada por n ceros, esto implica que las columnas de H no pueden ser linealmente independientes, porque es posible obtener el vector nulo haciendo una combinación no trivial de ellas.

Más aún, miremos los lugares en los que una palabra $\mathbf{c}$ del código tiene componentes distintas de cero. Entonces el subconjunto de columnas de H que corresponde a esos lugares es linealmente dependiente, porque hay una combinación lineal de ellas, con coeficientes no nulos, que es igual a cero. Podemos olvidarnos de las restantes columnas, porque quedan multiplicadas por 0 al hacer el producto de H por $\mathbf{c}^t$. Si recordamos que habíamos definido el *peso* de una palabra del código como la cantidad de componentes no nulas, podemos extraer la siguiente conclusión: si en el código hay una palabra con peso s, entonces existe al menos un conjunto formado por s columnas de la matriz de control que es linealmente dependediente.

Podemos ir también en la otra dirección. Si un subconjunto

$$\{\mathbf{h}_{r_1}, \ldots, \mathbf{h}_{r_s}\}$$

formado por s columnas de la matriz de control H es linealmente dependiente, podemos encontrar escalares $a_{r_1}, \ldots, a_{r_s}$ (no todos nulos) tales que

$$a_{r_1}\,\mathbf{h}_{r_1} + \cdots + a_{r_s}\,\mathbf{h}_{r_s} = \mathbf{0}^t\,.$$

Pero eso quiere decir que la palabra $\mathbf{a}$ que está formada por los a_{r_j} en las posiciones r_j (y cero en el resto) cumple que

$$H\,\mathbf{a}^t = \mathbf{0}^t\,,$$

así que pertenece al código. Además su peso es $\leq s$ (el peor caso sería que todos los a_{r_j} fueran no nulos). Por lo tanto, si hay un subconjunto linealmente dependiente, formado por s columnas de la matriz de control, el código tiene al menos una palabra de peso menor o igual que s. También podemos formular las cosas de la siguiente manera: si el código no tiene ninguna palabra con peso menor que s entonces cualquier subconjunto formado por menos de s columnas de la matriz de control H debe ser linealmente independiente.

Los comentarios de los últimos párrafos se resumen en nuestro próximo teorema.

Teorema 3 *La distancia mínima $d_{\mathcal{C}}$ de un código lineal $\mathcal{C}$ con matriz de control H coincide con el menor cardinal de un conjunto de columnas linealmente dependientes en H.*

DEMOSTRACIÓN. Sea $\mathbf{c} = (c_1, \ldots, c_n)$ una palabra del código que tenga peso $d_{\mathcal{C}}$. Sabemos que existe porque ésta es la distancia mínima y el código es lineal. Entonces, tal como observábamos antes, el conjunto de columnas $\{\mathbf{h}_j \,:\, c_j \neq 0\}$, que corresponde a las entradas no nulas de la palabra $\mathbf{c}$, es linealmente dependiente. El cardinal de este conjunto es $d_{\mathcal{C}}$.

Por otra parte, no puede haber un conjunto de columnas linealmente dependientes con cardinal menor, porque entonces el código tendría una palabra con peso menor que $d_{\mathcal{C}}$, en contradicción con el hecho de que $d_{\mathcal{C}}$ es la distancia mínima. □

Como aplicación de este teorema, podemos calcular las distancias mínimas de algunos de los ejemplos vistos anteriormente.

En el ejemplo del control de paridad, la matriz era

$$H = \begin{pmatrix} 1 & 1 & \cdots & 1 \end{pmatrix},$$

que, por supuesto, tiene dos columnas linealmente dependientes, y verificamos con esto nuestro cálculo previo de $d_{\mathcal{C}} = 2$.

Para el código $[7, 4, 3]$ la matriz de control era

$$H = \begin{pmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{pmatrix}$$

observamos que no hay columnas repetidas (así que no hay pares linealmente dependientes), pero sí hay tríos linealmente dependientes: por ejemplo, al sumar la primera y la segunda columna, obtenemos la cuarta. Así que $d_{\mathcal{C}} = 3$, como habíamos obtenido con el cálculo del peso de las palabras del código. Vale la pena observar que la matriz H contiene todas las posibles columnas de longitud 3 formadas por ceros y unos, salvo la que sólo tiene ceros. Esta observación será la clave para la construcción sistemática de una familia de códigos perfectos conocidos como códigos de Hamming, y será también de gran ayuda para descodificar.

Ejercicio 4.34 Verificar con la matriz de control todos los cálculos de distancias mínimas hechos hasta el momento.

Habíamos observado en el ejercicio 4.8 de la página 42 que un código con distancia mínima $d_{\mathcal{C}}$ era capaz de corregir hasta $d_{\mathcal{C}} - 1$ borrones. El objetivo de nuestro próximo ejercicio es poner en práctica esta capacidad de corrección en el caso de los códigos lineales.

Ejercicio 4.35 CORRECCIÓN DE BORRONES

1. Codifiquemos palabras de longitud 3 como palabras de longitud 4 agregando un dígito de control de paridad. Determinar, si es posible, cuál fue la palabra codificada para cada uno de los siguientes casos:

$$(1, 0, \ , 1), \quad (1, \ , \ , 1).$$

2. Hacer lo mismo que en la parte anterior para el código [7,4,3] que fue introducido en el ejemplo 4.20, y las palabras

$$(1, \ , 0, \ , 1, 1, 1), \quad (\ , 1, \ , 1, 1, 1, \ ).$$

3. Dar un procedimiento general para corregir hasta $d_{\mathcal{C}} - 1$ borrones en palabras de un código con distancia mínima $d_{\mathcal{C}}$, y explicar por qué funciona el procedimiento.

4.3.5 Códigos de Hamming

El resultado anterior permite construir de manera sistemática una familia de códigos lineales binarios (basados en la aritmética de $\mathbb{F}_2$) cuya distancia mínima es 3. Es decir, códigos que corrigen un error. Recordemos que construir un código lineal $\mathcal{C}$ no es otra cosa que dar un subespacio de dimensión k en $\mathbb{F}_2^n$, para algún valor de n y k. Utilizaremos luego este subespacio para representar palabras del diccionario original $\mathcal{P} = \mathbb{F}_2^k$.

Caracterizaremos los $[n, k]$ códigos de nuestra construcción por medio de su matriz de control H, que será de dimensiones $(n - k) \times n$. Como queremos que la distancia mínima sea 3 buscaremos que haya algún conjunto de tres columnas de H que sea linealmente dependiente, y que ningún par

de columnas forme un conjunto linealmente dependiente. Esto último es especialmente fácil de conseguir con la aritmética de $\mathbb{F}_2$: basta con que no esté el vector nulo de $\mathbb{F}_2^{n-k}$ entre las columnas de H, ni haya dos columnas repetidas. La condición de que haya tres columnas que formen una familia linealmente dependiente se consigue colocando en la matriz de control *todas las columnas no nulas* de longitud $n-k$. Esta elección de H tiene algunas ventajas: una vez que hemos fijado el valor de $n-k$ es la que produce el código que tiene mayor cantidad de palabras y menor redundancia posible (es decir, es el que requiere agregar menos información para asegurarse de corregir un error). Consideraremos estas afirmaciones en los próximos párrafos. Para nuestros cálculos llamaremos s a la diferencia $n-k$. Justifiquemos primero algunas de las cosas que hemos dicho antes, resolviendo el siguiente ejercicio sobre las combinaciones lineales de vectores de $\mathbb{F}_2^s$.

Ejercicio 4.36

1. Mostrar que un par de vectores diferentes y no nulos de $\mathbb{F}_2^s$ forma una familia linealmente independiente.

2. Mostrar que la suma de dos vectores diferentes y no nulos de $\mathbb{F}_2^s$ da lugar a un tercer vector que es diferente de los dos sumandos, y que también es distinto del vector nulo del espacio. Concluir que en el conjunto formado por los 2^s-1 vectores no nulos de $\mathbb{F}_2^s$ hay familias linealmente dependientes formadas por tres vectores.

3. Mostrar que si tres vectores $\mathbf{v}_1$, $\mathbf{v}_2$ y $\mathbf{v}_3$ son distintos entre sí, distintos del vector nulo, y forman una familia linealmente dependiente, entonces cualquiera de ellos es igual a la suma de los otros dos.

El parámetro n del código es igual a la longitud de sus palabras. Y coincide con el número de columnas en la matriz de control. Las columnas de esta matriz tienen s elementos, que tendremos que escoger en el conjunto $\{0,1\}$. Para cada lugar hay dos posibilidades, así que hay en total 2^s columnas posibles. Como no podemos poner una columna formada por ceros en la matriz de control debemos escoger

$$n = 2^s - 1.$$

Observemos que este es el número máximo de columnas que puede tener la matriz de control si queremos que la distancia mínima del código sea 3. Una matriz de s filas con más columnas necesariamente tendrá columnas repetidas, o, aún peor, una columna de ceros. Por lo tanto, una vez fijado el valor de s ésta es la mayor longitud posible para las palabras que es compatible con la condición de que el código corrija un error. Dado que $s = n-k$ tendremos que $k = n-s$. Por lo tanto, para s fijo, escogiendo el mayor valor de n posible también nos aseguramos de elegir el k más

grande que permite obtener un código con las propiedades deseadas. Ahora k resulta muy fácil de calcular, en efecto

$$k = n - (n - k) = n - s = 2^s - s - 1.$$

Analicemos ahora la tasa de transmisión que tienen estos códigos que corrigen un error. Esta tasa es igual a

$$R_{\mathcal{C}} = k/n = 1 - s/n.$$

Vemos entonces que, para un valor de s fijado, la elección que conduce a una tasa de transmisión mayor es la que corresponde a tomar el mayor valor de n posible, y estamos agregando así menos información redundante para alcanzar nuestro objetivo de corregir un error.

En resumen, hemos construido un $[n, k, 3]$ código, con parámetros

$$n = 2^s - 1 \qquad \text{y} \qquad k = 2^s - s - 1\,.$$

Para cada valor de $s \geq 2$, llamaremos **s-códigos de Hamming binarios** a los códigos lineales con parámetros $[2^s - 1, 2^s - s - 1, 3]$: un subespacio de dimensión $2^s - s - 1$ del espacio de las palabras de longitud $2^s - 1$ con distancia mínima 3 (por tanto, corrigen un error). El del ejemplo 4.20 era el código de Hamming para $s = 3$, un $[7, 4, 3]$-código. Estos códigos de Hamming son perfectos, pues el número de palabras de que constan es

$$|\mathcal{C}| = 2^{2^s - s - 1}\,,$$

y, por tanto,

$$|\mathcal{C}| \left(\binom{2^s - 1}{0} + \binom{2^s - 1}{1} \right) = 2^{2^s - s - 1}\ (1 + 2^s - 1) = 2^{2^s - 1}\,,$$

se alcanza la igualdad en la cota de Hamming.

La siguiente tabla recoge los parámetros de los primeros códigos de Hamming binarios:

s	$n = 2^s - 1$	$k = 2^s - s - 1$	$\lvert\mathcal{C}\rvert = 2^{2^s - s - 1}$	$R_{\mathcal{C}} = k/n$
2	3	1	2^1	$1/3 = 0.33$
3	7	4	2^4	$4/7 = 0.57$
4	15	11	2^{11}	$11/15 = 0.73$
5	31	26	2^{26}	$26/31 = 0.84$
6	63	57	2^{57}	$57/63 = 0.90$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

Observamos que la redundancia $1 - R_{\mathcal{C}}$ disminuye cuando aumentamos el valor de s. En la sección 5 hemos incluido la comparación de la eficacia en la corrección de errores (medida en términos probabilísticos) que tienen estos códigos.

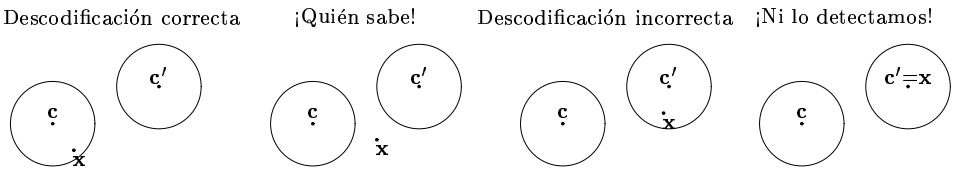
4.3.6 Descodificando cuando hay errores

Hasta el momento hemos fabricado algunos códigos, anunciando que servirían para corregir errores en la transmisión de información, y analizado el proceso de descodificación cuando tenemos que interpretar una palabra del código. Estudiaremos ahora el caso general, en que nos veremos enfrentados con la necesidad de interpretar palabras que no están en el código, o reconstruir una palabra que tiene borrones.

Repasemos una vez mas todo el proceso, Supongamos entonces que hemos codificado una palabra $\mathbf{p}$ del diccionario original $\mathcal{P} \subseteq \mathbb{F}_2^k$ como una palabra $\mathbf{c}$ del código, $\mathcal{C} \subset \mathbb{F}_2^n$. Este código es lineal $[n, k]$, y digamos que su distancia mínima es $d_{\mathcal{C}} = 2e + 1$, así que corrige hasta e errores. Enviamos la palabra $\mathbf{c}$ por el canal y recibimos la palabra de $\{0, 1\}^n$ siguiente:

$$\mathbf{x} = \mathbf{c} + \mathbf{e}\,,$$

donde $\mathbf{e}$ es el (hipotético) error que añade el canal. El criterio de descodificación que emplearemos es el del *vecino más próximo*, que consiste, tal como hemos mencionado anteriormente, en buscar la palabra del código $\mathcal{C}$ que está más cerca de $\mathbf{x}$. Si en el proceso se han cometido e errores como mucho, es decir, si $\omega(\mathbf{e}) \leq e$, entonces este proceso recupera la palabra $\mathbf{c}$: $d(\mathbf{x}, \mathbf{c}) = \omega(\mathbf{e}) \leq e$, y $\mathbf{c}$ es la única palabra del código con esa propiedad. Si se han cometido más de e errores, pero no más de $d_{\mathcal{C}}$, entonces detectaremos el error, aunque no sabremos corregirlo en general. Y si se han producido más de $d_{\mathcal{C}}$ errores, el proceso de descodificación nos dará, seguramente, un resultado equivocado; de hecho, podría ocurrir que ni siquiera detectáramos el error. Los siguientes esquemas describen las distintas posibilidades (las bolas tiene radio e, $\mathbf{c}$ fue la palabra transmitida y $\mathbf{x}$ es la recibida):



Una implementación de esta forma de descodificar que es válida para cualquier código es la siguiente:

1. calcular todas las distancias de $\mathbf{x}$ a las palabras del código;
2. Descodificar la palabra $\mathbf{x}$ (esto es, asignarle una palabra del código) como la palabra de $\mathcal{C}$ más cercana a ella.

Sin embargo, en un código dotado de cierta estructura, como son los códigos lineales que estamos examinando, este procedimiento resulta demasiado ingenuo y antieconómico. A continuación presentamos una forma más eficiente de realizar la descodificación, que está basada en la matriz de control.

Sabemos que si $\mathbf{c}$ es una palabra del código, entonces

$$H\,\mathbf{c}^t = \mathbf{0}^t\,,$$

donde $\mathbf{0}$ tiene longitud $n-k$. Pero si aplicamos la matriz H a una palabra $\mathbf{x}$ que no sea del código, obtendremos un vector no nulo a la derecha: este vector que delata el error, y, en algún sentido, permite indentificarlo, ocupará un lugar destacada en el proceso de descodificación. Por eso introducimos la siguiente definición.

Definición 3 *Dada una palabra* $\mathbf{x} \in \{0,1\}^n$*, su* ***síndrome****,* $s(\mathbf{x})$*, es el vector de* $\{0,1\}^{n-k}$ *dado por*

$$H\,\mathbf{x}^t = s(\mathbf{x})^t\,.$$

Como comentábamos antes, si $s(\mathbf{y}) = \mathbf{0}$, entonces $\mathbf{y}$ pertenece al código (y viceversa). Además, si tras el proceso de transmisión de una palabra $\mathbf{c}$ hemos recibido $\mathbf{x} = \mathbf{c} + \mathbf{e}$, entonces, el síndrome de $\mathbf{x}$ es

$$s(\mathbf{x}) = H\mathbf{x}^t = H(\mathbf{c}+\mathbf{e})^t.$$

Pero el vector que se obtiene de trasponer la suma de $\mathbf{c}$ y $\mathbf{e}$ es el mismo que resulta de trasponer y luego sumar, de modo que

$$s(\mathbf{x}) = H(\mathbf{c}^t + \mathbf{e}^t) = H\mathbf{c}^t + H\mathbf{e}^t.$$

Pero los dos sumandos en el último miembro de esta serie de igualdades son conocidos. El primero es nulo, porque $\mathbf{c}$ está en el código. El segundo es el síndrome del error $\mathbf{e}$. Concluimos entonces que

$$s(\mathbf{x}) = s(\mathbf{e})\,.$$

En otras palabras, el síndrome de $\mathbf{x}$ coincide con el síndrome del error, así que obtener el síndrome del error es inmediato (basta medir el síndrome de la palabra recibida).

Naturalmente, varios errores diferentes pueden dar lugar al mismo síndrome, como veremos en el siguiente ejemplo

Ejemplo 4.37 La matriz de control del código de Hamming $[7,4,3]$ es

$$H = \begin{pmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{pmatrix},$$

de modo que el síndrome del vector $(1,1,1,0,0,0,1)$ es $(0,1,1)$. Pero este último vector es también el síndrome de $\mathbf{e_2} = (0,1,0,0,0,0,0)$, y en realidad es el síndrome del vector que resulta de sumar $\mathbf{e_2}$ a cualquier palabra del código. ♣

Pero nuestro criterio de descodificación implica que debemos asumir que de todos los errores posibles el error cometido es el más pequeño posible. Dado que ya sabemos como calcular el síndrome del error, nos bastará entonces escoger el vector más chico de entre los que tienen el síndrome dado.

Ejemplo 4.38 En el ejemplo anterior, el vector más pequeño con síndrome $(0,1,1)$ es $\mathbf{e_2}$. Este vector tiene peso 1, de modo que no puede haber otro que sea estrictamente más pequeño. Por otra parte, los vectores que tienen peso 1 son solamente los $\mathbf{e}_i$, $i = 1,\ldots,7$ de la base canónica de $\{0,1\}^7$. Cada uno de ellos, al ser multiplicados a la derecha por H produce, como resultado la i-ésima columna de H. Por lo tanto, el único que tiene síndrome $(0,1,1)$ es el $\mathbf{e_2}$, y es, en consecuencia, el de menor peso entre todos los que tienen este síndrome. El criterio de vecino más próximo nos lleva entonces a considerar que el error cometido en cualquier vector con síndrome $(0,1,1)$ es $\mathbf{e_2}$. Entonces, al recibir el vector $(1,1,1,0,0,0,1)$ del ejemplo anterior debemos intepretar que se ha intentado transmitir el vector $(1,0,1,0,0,0,1)$ del código. ♣

Para seguir adelante conviene introducir un poco de jerga. Para cada síndrome $\mathbf{s}$ consideraremos el conjunto de todas las palabras de $\{0,1\}^n$ con síndrome $\mathbf{s}$ al que llamaremos $\mathcal{E}_\mathbf{s}$. Nos referiremos a esta familia $\mathcal{E}_\mathbf{s}$ como *la clase de palabras con síndrome* $\mathbf{s}$.

Ejemplo 4.39 Si $\mathbf{s} = \mathbf{0}$ entonces $\mathcal{E}_\mathbf{s} = \mathcal{C}$. En otros términos, la clase de las palabras con síndrome nulo es el propio código. ♣

El siguiente ejercicio aclara un poco la estructura de las operaciones que estamos haciendo, aunque no sacaremos luego mayor partido de ello.

Ejercicio 4.40

1. Mostrar que la relación $\mathbf{u} \sim \mathbf{v}$ si $s(\mathbf{u}) = s(\mathbf{v})$ es una relación de equivalencia, y que las clases que esta relación de equivalencia definen son justamente los conjuntos $\mathcal{E}_\mathbf{s}$.
2. Mostrar que $\mathbf{u} \sim \mathbf{v}$ si y sólo si $\mathbf{u} - \mathbf{v} \in \mathcal{C}$
3. Mostrar que cada conjunto $\mathcal{E}_\mathbf{s}$ puede escribirse en la forma

$$\mathcal{E}_\mathbf{s} = \{\mathbf{x} + \mathbf{c} \,:\, \mathbf{c} \in \mathcal{C}\},$$

   donde $\mathbf{x}$ es cualquier pálabra con síndrome $\mathbf{s}$.
4. Concluir que cada clase tiene exactamente 2^k elementos, y que hay 2^{n-k} clases.

Tenemos entonces que cada clase $\mathcal{E}_\mathbf{s}$ contiene todos los posibles errores que podrían llevar a producir el síndrome $\mathbf{s}$, y para descodificar siguiendo el criterio del vecino más próximo nos interesa identificar al más pequeño

de estos errores. Entonces, si una clase tiene un *único* elemento de peso mínimo, éste será el llamado **líder** de la clase. Si observamos una palabra con síndrome $\mathbf{s}$, y la clase $\mathcal{E}_{\mathbf{s}}$ tiene líder $\mathbf{e}$ asumiremos que el error cometido es justamente $\mathbf{e}$. Al ser $\mathbf{e}$ el líder cualquier otro error que arrojara el mismo síndrome tendría peso mayor, y si lo aceptáramos como error en vez de $\mathbf{e}$ no estaríamos descodificando siguiendo el criterio del vecino más próximo.

Resumiremos este paso del procedimiento de descodificación en nuestra siguiente proposición.

Proposición 1 *Sea* $\mathbf{y}$ *una palabra de* $\{0,1\}^n$ *con síndrome* $\mathbf{s}$. *Supongamos que la clase* $\mathcal{E}_{\mathbf{s}}$ *tiene líder* $\mathbf{e}$. *Entonces*

$$\mathbf{x} = \mathbf{y} - \mathbf{e}$$

es la única palabra del código $\mathcal{C}$ *que satisface*

$$d(\mathbf{y}, \mathbf{x}) = \min_{\mathbf{c} \in \mathcal{C}} d(\mathbf{y}, \mathbf{c}).$$

DEMOSTRACIÓN. El síndrome de la diferencia entre $\mathbf{y}$ y cualquier palabra $\mathbf{c}$ del código es

$$s(\mathbf{y} - \mathbf{c}) = s(\mathbf{y}) - s(\mathbf{c}) = \mathbf{s} - \mathbf{0} = \mathbf{s}.$$

Por lo tanto, estas diferencias están en la clase $\mathcal{E}_{\mathbf{y}}$, y todas tienen un peso estrictamente mayor que $\mathbf{e}$, que es el líder de la clase. Por otra parte, el síndrome de $\mathbf{x}$ es

$$s(\mathbf{x}) = s(\mathbf{y}) - s(\mathbf{e}) = \mathbf{s} - \mathbf{s} = \mathbf{0},$$

de modo que efectivamente $\mathbf{x}$ está en el código $\mathcal{C}$. Como

$$\mathbf{e} = \mathbf{y} - \mathbf{x}$$

encontramos entonces que $\mathbf{x}$ es la palabra del código que está a distancia mínima de $\mathbf{y}$, y además es la única palabra del código a esa distancia. □

Ejemplo 4.41 SÍNDROMES Y LÍDERES PARA LOS CÓDIGOS DE HAMMING BINARIOS

En el caso de los códigos de Hamming binarios es muy sencillo cálcular los líderes de todas las clases, y la forma de hacerlo es una extensión de la que empleamos en el ejemplo 4.38. En efecto, en la matriz de control H de un código de Hamming están todas las columnas de $\mathcal{F}_2^s$, salvo la nula, que son los posibles síndromes de las listas que caen fuera del código. Ahora, la i-ésima columna de la matriz H se obtiene multiplicando a la derecha por una columna que tiene un 1 en el lugar i, y ceros en todas las demas entradas. Este vector esta a distancia 1 del origen, y es el único entre los vectores de peso uno que produce ese síndrome (un vector de peso uno diferente tiene como síndrome una columna distinta de la matriz H, que no puede coincidir con la i-ésima por la construcción de H). Por lo tanto, es el más cercano al origen de todos los de su clase: su líder.

Ejercicio 4.42 Se emplea el código de Hamming $[7,4,3]$ para codificar palabras de longitud 4 por medio de palabras de longitud 7.

1. Si se reciben las palabras

$$(1,1,1,1,0,0,0), \quad (1,0,1,0,1,0,1),$$

¿cuáles son las palabras de longitud 7 que debemos considerar como el verdadero contenido del mensaje?

2. ¿Cuáles son las palabras originales, de longitud 4, que se nos quizo mandar? Naturalmente, la respuesta a esta pregunta depende de la codificación empleada. Responderla para:

   (a) una codificación sistemática;

   (b) una codificación que use la matriz (6) de la página 53. ♣

Hemos visto entonces que en los códigos de Hamming cada síndrome da lugar a una clase que tiene un líder. Pero puede ocurrir también que haya clases con más de un elemento de longitud mínima, lo que hace que el líder no exista y se creen situaciones en las que no se puede descodificar. Veamos un ejemplo a continuación.

Ejemplo 4.43 Tomemos el $[7,2]$ código

$$\mathcal{C} = \left\{ \begin{array}{ll} (0,0,0,0,0,0,0), & (1,1,1,0,0,0,0), \\ (0,0,0,1,1,1,1), & (1,1,1,1,1,1,1) \end{array} \right\}$$

Ejercicio 4.44

1. Calcular la distancia mínima del código y una matriz de control. Hallar todos los síndromes que corresponden a palabras que difieren exactamente en una posición de una palabra del código, las clases asociadas a estos síndromes y sus líderes.

2. En una transmisión recibimos $(1,1,1,1,1,0,0)$. Hallar dos palabras del código a distancia 2 de esta palabra, y verificar que no hay ninguna a distancia 1. Calcular su síndrome y la clase correspondiente a este síndrome. Mostrar que la clase no tiene líder.

3. Se reciben las palabras

$$(1,1,1,0,0,1,0), \quad (1,1,0,1,0,0,0).$$

¿Cómo deben ser descodificadas?

4. Mostrar que si se produce un error en la transmisión el código puede corregirlo. Pero que si se producen dos puede ocurrir cualquiera de las siguientes tres cosas:

   (a) que el código corrija correctamente y devuelva el mensaje original;

   (b) que el código nos haga interpretar una palabra equivocada;

(c) que no sepamos cómo descodificar.

Construir ejemplos para las tres situaciones. ♣

Hemos visto entonces ejemplos en los que todas las clases tienen líderes, y un ejemplo en que hay clases sin líder. El siguiente resultado asegura la existencia de líderes bajo ciertas condiciones.

Proposición 2 *Si un código $\mathcal{C}$ tiene distancia mínima $d_{\mathcal{C}} = 2e+1$, entonces ninguna clase $\mathcal{E}$ puede tener más de un elemento con peso $\leq e$.*

DEMOSTRACIÓN. Si la clase es la correspondiente al código $\mathcal{C}$, entonces el elemento $\mathbf{0}$ de esta clase tiene peso cero, y los demás tienen peso mayor o igual que $2e+1$. Consideremos entonces cualquier otra clase, y dos elementos suyos, $\mathbf{x}$ e $\mathbf{y}$, ambos de peso $\leq e$. Entonces la palabra $\mathbf{x}-\mathbf{y}$ pertenece al código, y se cumple que

$$\omega(\mathbf{x}-\mathbf{y}) = d(\mathbf{x},\mathbf{y}) \leq d(\mathbf{x},\mathbf{0}) + d(\mathbf{y},\mathbf{0}) = \omega(\mathbf{x}) + \omega(\mathbf{y}) \leq 2e < d_{\mathcal{C}}\,,$$

recordando la desigualdad triangular que cumplía la distancia de Hamming. Como $\omega(\mathbf{x}-\mathbf{y}) = d(\mathbf{x},\mathbf{y})$ y la distancia de Hamming es, efectivamente, una distancia, sólo queda la posibilidad de $\mathbf{x}-\mathbf{y} = \mathbf{0}$. Así que ambos elementos son el mismo, en realidad. □

El resultado que acabamos de probar muestra que las clases que corresponden a palabras con no muchos errores, no más de e errores, tienen líder. En consecuencia, podremos descodificar estas palabras sin ninguna ambigüedad.

Estamos en condiciones ahora de resumir todo el proceso de descodificación. Supongamos que hemos recibido una lista $\mathbf{y}$: descodificarla supone encontrar la palabra del código que esté más cercana a ella, y la descodificación será posible si esta palabra es única. Tendríamos que buscar la palabra $\mathbf{c} \in \mathcal{C}$ (si es que hay una sola) para la que se alcance el mínimo.

Hemos visto que hacer esto es equivalente a identificar el líder en la clase de $\mathbf{y}$. Basta, pues, tener la lista de las clases y sus líderes para poder descodificar. Detallemos el algoritmo: dado un código lineal $[n,k]$, construimos una tabla con dos columnas, una para el síndrome $\mathbf{s}$ y otra para el líder (si existe), y 2^{n-k} filas, una por cada clase $\mathcal{E}_{\mathbf{s}}$. Cada fila representa entonces a una clase $\mathcal{E}$: en la primera columna escribiremos el síndrome de un elemento cualquiera del bloque (recordemos que el síndrome era común a todos ellos), y en la segunda incluiremos el líder de esa clase, si es que existe. Si ahora recibimos una lista $\mathbf{y}$,

1. calculamos $s(\mathbf{y})$ y lo localizamos en la columna de síndromes. Ya sabemos a qué bloque pertenece $\mathbf{y}$.

2. Si la clase seleccionada no posee líder, entonces no vamos a poder descodificar. Pero si sí tiene líder, digamos $\mathbf{e}$, entonces decidimos que $\mathbf{e}$ es el error cometido, así que descodificamos a $\mathbf{c} = \mathbf{y} - \mathbf{e}$.

3. La palabra $\mathbf{c}$ está en el código. Por lo tanto proviene de alguna palabra $\mathbf{p}$ a través de la función de codificación f. En otras palabras, existe una palabra $\mathbf{p}$ en nuestro diccionario original tal que $\mathbf{c} = f(\mathbf{p})$. Recuperamos esta palabra $\mathbf{p}$ deshaciendo la codificación por medio de la aplicación de f^{-1}, como $\mathbf{p} = f^{-1}(\mathbf{c})$. Si la codificación había sido sistemática, este último paso es sencillo: basta leer las primeras coordenadas de la palabra código (el resto era la información redundante). Si no lo estaba, es posible calcular f^{-1} y almacenar esta información en una matriz[53].

Las ventajas del proceso son obvias: dado el código, la tabla se genera una vez y sirve para cualquier descodificación. Todo el proceso de codificación-descodificación puede resumirse en la siguiente figura:

$$\mathbf{p} \xrightarrow{f} \mathbf{c} \overset{\text{canal}}{\rightsquigarrow} \mathbf{y} \overset{\text{interpretamos}}{\longrightarrow} \mathbf{c} \xrightarrow{f^{-1}} \mathbf{p}$$

Hay que recordar que no siempre vamos a obtener la palabra original $\mathbf{p}$. A veces el criterio de vecino más próximo no decide cuál es la palabra $\mathbf{c}$ que debemos considerar. Otras veces, si se han producido muchos errores, el criterio nos podría llevar a una palabra del código que no fuera la emitida; tras aplicar f^{-1} llegaríamos a una palabra distinta de la $\mathbf{p}$ original. Pero en fin, el código corrige lo que corrige, y no más.

Ejercicio 4.45 Calcular la tabla de síndromes y líderes para el código del ejercicio 4.22. Escoger una palabra, y agregarle dos errores de modo que:

1. el criterio de corrección basado en el vecino más próximo no permita decidir como corregir;
2. el criterio induzca a tomar una decisión errónea.

Ejercicio 4.46 Elaborar una tabla de síndromes y líderes para el código de repetición que consiste en repetir tres veces palabras de dos dígitos.

Ejercicio 4.47

1. Mostrar que hay una forma muy natural de poner en correspondencia todas las columnas de la matriz de control de un código de Hamming binario con parámetro s (es decir, los $2^s - 1$ vectores no nulos de $\mathbb{F}_2^s$) con los primeros $2^s - 1$ números naturales. Sugerencia: recurrir a la representación binaria de los números naturales.
2. Utilizar la correspondencia de la parte anterior para escribir la matriz de control de los códigos de Hamming de una forma que permita describir muy sencillamente el proceso de descodificación. Sugerencia: para descodificar las palabras que tienen un error hay que identificar el lugar en que éste se cometió.

[53] Esta matriz será una inversa a la derecha para la matriz de codificación G.

3. Para el código de Hamming $[7, 4, 3]$ construir una matriz de control y una matriz generatriz que hagan los procesos de codificación y descodificación tan fáciles como sea posible.

Ejercicio 4.48 CORRECCIÓN DE ERRORES Y BORRONES
Consideraremos un código lineal con función de codificación

$$f : \mathbb{F}_2^k \to \mathcal{C} \subset \mathbb{F}_2^n,$$

donde el código $\mathcal{C}$ tiene distancia mínima $d_{\mathcal{C}}$, y números naturales b y r tales que $b + 2r < d_{\mathcal{C}}$.

1. Definimos una función

   $$\tilde{f} : \{0, 1\}^k \to \mathcal{C} \subset \{0, 1\}^{n-b}$$

   que se construye codificando con f, y luego eliminando b entradas (siempre las mismas) de los vectores del código $\mathcal{C}$ resultantes. Mostrar que $\tilde{f}$ es una función inyectiva, por lo que es una función de codificación válida sobre un nuevo código $\mathcal{C}'$. Dar una cota inferior para la distancia mínima de este nuevo código.

2. Hemos visto en el ejercicio 4.8 de la página 42 que el código original f permite recuperar correctamente una palabra que fue transmitida con e errores y b borrones. Hallar una manera eficiente de recuperarla (sugerencia: usar el código $\mathcal{C}'$ que se obtiene eliminando las b columnas con borrones).

3. Se transmite cada palabra de 2 dígitos repetida 5 veces según el esquema $(a, b, a, b, \ldots, a, b)$. Se recibe

   $$(0, 0, 1, 0, \ , 0, 1, \ , 1, 0).$$

   Determinar la palabra (a, b) enviada.

Para nuestro próximo ejercicio simularemos un canal con ruido. Dada una lista de ceros y unos generaremos otra lista según el siguiente procedimiento: miramos el primer carácter de la lista y arrojamos un dado. Si el resultado del dado es menor que 6 copiamos el carácter tal cual. Pero si el resultado es 6 introducimos un "error", poniendo cero si había un uno, y uno si había un cero[54]. De este modo hemos generado el primer dígito de la nueva lista. Repetimos el procedimiento para todos los dígitos de la lista original, hasta "completar la transmisión".

Ejercicio 4.49 Tomar las frases que se tradujeron en listas de ceros y unos en el ejercicio 3.3, página 21. Transmitirlas por el procedimiento que acabamos de describir y descodificarlas. Hacerlo de varias maneras:

1. transmitiendo la lista de ceros y unos sin ningún tipo de procesamiento;

2. empleando un código de Hamming $[7, 4, 3]$ para hacer las transmisión;

[54] Nótese que éste es un canal que se "equivoca" con probabilidad 1/6.

3. usando el código de repetición $[12, 4, 3]$;
4. usando el código $[5, 2, 3]$ del ejercicio 4.22;
5. con un código de Hamming $[15, 11, 3]$ (que habrá que construir).

Comparar los resultados obtenidos y el tiempo que llevan los procesos de codificación y descodificación para los distintos procedimientos adoptados.

Ejercicio 4.50 Repetir el ejercicio anterior, pero emplear dados de 8 caras y de 20 caras para simular los errores (o generar otros procedimientos que introduzcan una probabilidad de equivocarse en cada carácter igual a $1/8$ y $1/20$).

En la sección 5 introduciremos un sencillísimo modelo probabilístico para la transmisión de información a través de un canal con ruido, y estudiaremos la performance de álgunos códigos correctores en este contexto.

5 Breves consideraciones probabilísticas

La necesidad de construir códigos que detecten y corrijan errores se debe, por supuesto, a que se pueden cometer errores en la transmisión de la información. La aparición de errores es un fenómeno esencialmente aleatorio. Es prácticamente imposible predecir con exactitud cuándo se va a producir un error, o que lugar de la información se va a ver afectado. En cambio, los errores tienen ciertas regularidades estadísticas que permiten introducir *modelos probabilísticos* para estudiar la fiabilidad de los procedimientos de transmisión de información y la eficacia de nuestros códigos correctores de errores. En esta sección presentaremos un modelo sumamente simplificado para la transmisión de información a través de un canal.

Supondremos que tenemos un mensaje ya codificado, y queremos enviar a un receptor una serie de palabras código (listas de ceros y unos de longitud n) a través de un canal que tiene las siguientes características:

- Es un canal **binario**: toma listas de ceros y unos y hace llegar listas del mismo tipo[55].

- También supondremos que el canal no tiene pérdida de información, es decir, la cantidad de *bits* que entran coincide con la cantidad que sale (el canal lee listas de longitud n y devuelve listas del mismo tipo).

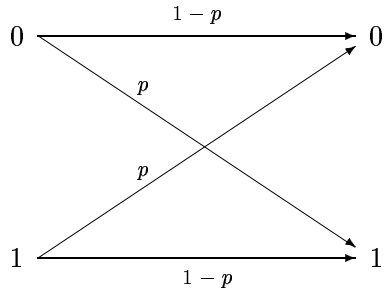
- Pero el canal tiene cierto **ruido**, cierta probabilidad de transmitir con errores. Nuestro canal es binario, así que deberíamos conocer la probabilidad de cada uno de los sucesos posibles: que transmita correctamente un 0 o un 1, que cambie un 0 por un 1 o un 1 por un 0. Esta información se puede recoger en una matriz 2×2,

$$\begin{pmatrix} prob(0 \to 0) & prob(1 \to 0) \\ prob(0 \to 1) & prob(1 \to 1) \end{pmatrix}$$

  Las entradas de cada columna de la matriz han de sumar 1, la probabilidad total[56]. Aquí supondremos que el canal es **simétrico**; es decir, es igualmente probable cambiar un 0 por 1 que un 1 por un 0. Esto supone que un único número, p, la probabilidad de digamos cambiar un 0 por un 1, describe todo el sistema:

[55] Vale la pena insistir en que estamos presentando un *modelo* para el funcionamiento del canal, y que este modelo reproducirá algunos aspectos de su funcionamiento pero no otros. Por ejemplo, un error habitual que el modelo que estemos manejando no tiene en cuenta es el de borrado, en el que un dígito, simplemente, no es transmitido.

[56] Este es un ejemplo de una **matriz de transición**, que describe como se redistribuye una población que está dividida en dos grupos (en este caso *ceros* y *unos*)



o, en términos de la matriz,

$$\begin{pmatrix} 1-p & p \\ p & 1-p \end{pmatrix}.$$

El número p es un parámetro del modelo, que debe ser estimado por algún procedimiento si se intenta aplicar el modelo a un sistema real. Podemos suponer que $p < 1/2$ (si fuera $p > 1/2$, simplemente cambiaríamos los papeles del 0 y el 1 a la derecha del esquema). Si $p = 1/2$, el canal es completamente aleatorio[57] y no podremos esperar recuperar ningún tipo de mensaje. En general asumiremos que el canal es bastante fiel, lo que se traduce en un valor de p próximo a 0. Y en la realidad esto ocurre. Si p es grande ¡el verdadero problema es el canal!

- Supondremos, por último, que el suceso en el que el canal cambia una cierta posición de la lista es completamente independiente de que cambie cualquiera otra[58].

Con todas estas hipótesis ya tenemos un modelo que describe la distribución de los errores en el mensaje. Vamos a estudiar ahora cómo afectan estos errores a la información enviada a través del canal que estamos modelando. En particular, mediremos cuan capaz es el proceso de codificación de reducir la probabilidad de cometer errores en la transmisión.

Comencemos por recordar cuáles son los ingredientes habituales de un proceso de codificación y transmisión: para formar nuestros mensajes utilizamos palabras de un diccionario $\mathcal{P} \subset \{0,1\}^k$, que codificamos con un código $\mathcal{C} \subset \{0,1\}^n$. Este código contiene $|\mathcal{C}|$ palabras y su distancia mínima es $d_{\mathcal{C}} = 2e + 1$. Agregamos ahora las hipótesis de que el canal utilizado para la transmisión de las palabras código es binario y simétrico, y p es la

[57] En el sentido de que lo que sale es independiente de lo que entra. En este canal es indiferente el mensaje que enviemos: el canal "sortea" una salida que no lo tiene en cuenta.

[58] Hay muchas situaciones en que ésta hipótesis no es fiel a la realidad. Por ejemplo, podemos tener una interferencia en el canal (por ejemplo, fenómenos meteorológicos) que provoque que una cierta porción del mensaje se vea más afectada por errores que el resto: son los llamados errores en ráfaga, o rachas de errores. Puede ocurrir también que el canal tenga una mayor probabilidad de cometer un error en, digamos, el primer dígito.

probabilidad de cometer un error (cambiar un 0 por un 1 o viceversa) en una posición de la lista. El número p es $< 1/2$, y generalmente será pequeño.

Ahora queremos enviar un mensaje que consta de un cierto número de palabras. Si lo hacemos con el diccionario original, cada palabra será una lista de k posiciones. Como hemos supuesto que los errores se producen en cada posición con arreglo a la probabilidad p e independientemente unos de otros, la probabilidad de transmitir correctamente una palabra será

$$prob(\text{transmisión correcta de una palabra}) = (1-p)^k$$

esto es, la probabilidad de que no se produzca ningún error en ninguna posición de la palabra. Ahora supongamos que utilizamos el código y calculemos la probabilidad de transmitir correctamente una palabra del código. Aun cometiendo hasta e errores, el criterio de vecino más próximo nos permitirá corregirlos y recuperar la original. También es posible[59] que, por casualidad, descodifiquemos correctamente alguna palabra que tenga más de e errores. En cualquier caso, si no hay mas de e errores siempre descodificaremos bien. Por lo tanto, todas las palabras con no más de e errores, y eventualmente algunas más, serán interpretadas como la palabra que se envió. Este hecho implica que la probabilidad de transmitir correctamente una palabra es mayor o igual que la de cometer no más de e errores. En un lenguaje un poco más formal escribimos

$$prob(\text{transmisión correcta de una palabra}) \geq prob(\text{cometer} \leq e \text{ errores}).$$

Es fácil evaluar la probabilidad de cometer exactamente j errores en la transmisión, donde j es un número cualquiera del conjunto $\{0, 1, \ldots, n\}$. Supongamos que además de fijar la cantidad de errores fijamos también los lugares en que tienen que producirse los errores —lo que determina inmediatamente que en las restantes posiciones no debe haber error alguno—. Entonces la probabilidad de obtener esta configuración es

$$(1-p)^{n-j}\, p^j.$$

Si sólo nos interesa la cantidad de errores, sin detenernos en el detalle de cuál es su posición en la lista, debemos tener en cuenta todas las posibles formas de poner j errores en una lista de n posiciones. El número de posibilidades es igual a $\binom{n}{j}$, es decir, al número de subconjuntos de j elementos contenidos en un conjunto de n elementos. Por lo tanto, la probabilidad de cometer exactamente j errores es

$$\binom{n}{j}(1-p)^{n-j}\, p^j. \tag{11}$$

[59] Esto depende del código que estemos utilizando, en los códigos perfectos no puede ocurrir. ¿Por qué?

Por último, la probabilidad de cometer hasta e errores se obtiene sumando (11) para los valores de j que son menores o iguales que e. En consecuencia

$$prob(\text{transmisión correcta de una palabra}) \geq \sum_{j=0}^{e} \binom{n}{j} (1-p)^{n-j} p^j.$$

Comparando los resultados de los dos cálculos que hemos hecho, observamos que el código mejora (reduce la probabilidad de error) la transmisión si

$$(1-p)^k < \sum_{j=0}^{e} \binom{n}{j} (1-p)^{n-j} p^j \ .$$

En los cálculos muchas veces nos interesará más que la transmisión de palabras aisladas la probabilidad de transmitir sin errores un mensaje que contiene h palabras. Como estamos suponiendo que los errores en una parte del mensaje son independientes de los errores en otras la probabilidad de transmitir correctamente h palabras es exactamente igual a la probabilidad de transmitir correctamente una palabra elevada a la h. En lo sucesivo utilizaremos este hecho reiteradamente.

Ejemplo 5.1 Queremos enviar un mensaje con 12 dígitos por un canal de $p = 0.01$. Supongamos primero que lo hacemos dígito a dígito. Entonces la longitud del mensaje es 12 y la probabilidad de transmitir correctamente es

$$(1-p)^{12} = (0.99)^{12} = 0.8864\,.$$

Casi un 12% de las veces tendremos algún error. Ahora supongamos que utilizamos un código de repetición: cada símbolo, sea un 0 o un 1, lo enviamos tres veces seguidas (recordemos que este código corrige un error). Ahora hemos introducido mucha redundancia (de hecho, la tasa de transmisión de este código es 1/3, lo que indica que sólo la tercera parte de lo que se envía es información, y el resto es redundancia agregada por la codificación), y el mensaje tiene longitud 36 El receptor interpretará correctamente cada palabra si no tiene más de un error. Por lo tanto, la probabilidad de transmisión correcta de *una* palabra es

$$\binom{3}{0}(1-p)^3 + \binom{3}{1}(1-p)^2 p = (0.99)^3 + 3\,(0.99)^2\,(0.01) = 0.999702.$$

Como el mensaje consta de 12 palabra la probabilidad de transmitirlo correctamente empleando este código de repetición es igual a

$$0.999702^{12} \approx 0.9964$$

por lo que ahora sólo el 0.36% de las ocasiones tendremos errores en la transmisión. Esto es, multiplicando por 3 el gasto que supone mandar el mensaje, reducimos en un factor 35 la probabilidad de error.

Ejercicio 5.2 Calcular la probabilidad de error si se emplea un código de repetición con palabras de longitud 5.

También podemos enviar nuestro mensaje empleando el código de Hamming $[7, 4, 3]$ longitud $k = 4$ en palabras de $\{0, 1\}^7$ y que corrige un error, agrupando nuestros 12 dígitos en 3 palabras de cuatro dígitos, y luego codificándolos en 3 bloques de 7 dígitos (este nuevo código tiene una tasa de transmisión igual a $4/7$). Y la probabilidad de que el mensaje sea interpretado correctamente es

$$\left((1-p)^7 + 7\,(1-p)^6\,p\right)^3 = 0.9939\,.$$

Doblamos (en realidad, un poquito más) la longitud del mensaje y reducimos los errores en un factor 20. ♣

Ejemplo 5.3 ANÁLISIS DE LA PERFORMANCE DE LOS CÓDIGOS DE HAMMING
Queremos transmitir un mensaje m que consta de 65208 dígitos binarios[60] por un canal con $p = 0.001$.

La primera estrategia sería enviar los 65208 bits directamente por el canal. La probabilidad de transmisión correcta resulta ser

$$prob(\text{transmisión correcta de } m) = (1-p)^{65208} \approx 10^{-28}\,.$$

Una probabilidad ínfima. Veamos el efecto de utilizar códigos de Hamming para realizar esta transmisión. Comencemos por un código de Hamming de parámetro $s = 2$. Para ello, cada dígito de m lo enviamos tres veces. El mensaje m se convierte, al codificarlo, en un mensaje c de 195.624 bits. Pero como esta codificación corrige hasta un error,

$$prob(\text{transmisión correcta de } c) = \left((1-p)^3 + 3\,p\,(1-p)^2\right)^{65208} = 0.822\,.$$

La mejora es brutal; aunque hemos tenido que pagar con triplicar el tamaño del mensaje.

Utilicemos ahora un código de Hamming $s = 3$. Para ello, partimos el mensaje m en 16302 bloques m_j de longitud 4. Cada bloque m_j lo codifico como una palabra c_j de longitud 7. Entonces,

$$prob(\text{transmisión correcta de } c_j) = (1-p)^7 + 7p\,(1-p)^6\,,$$

así que, para el mensaje c que consta de los sucesivos c_j,

$$prob(\text{transmisión correcta de } c) = \left((1-p)^7 + 7\,p\,(1-p)^6\right)^{16302} = 0.711\,.$$

[60] Hemos elegido 65208 porque es un múltiplo de todos los valores de k que vamos a utilizar en el ejemplo. Esta elección hace algo más sencillos los cálculos.

La mejora es menor (aunque sigue siendo notable); pero es que ahora el mensaje c es sólo 1.75 veces más largo que el original.

Si utilizamos el código de Hamming con $s = 4$, partimos m en 5928 palabras m_j de longitud 11. Y cada una de ellas la codificamos como una c_j de longitud 15. Así obtenemos que, para el mensaje codificado c,

$$prob(\text{transmisión correcta de } c) = \left((1-p)^{15} + 15\,p\,(1-p)^{14}\right)^{5928} = 0.539\,.$$

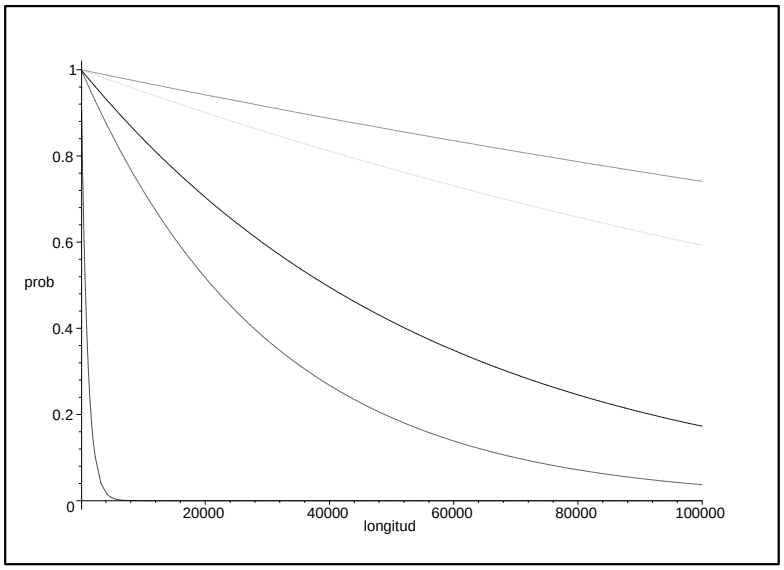
Ahora las palabras que transmitimos son 1.36 veces más largas. Para el Hamming con $s = 5$, la transmisión sería 1.19 veces más larga, y la probabilidad de enviar correctamente el mensaje sería

$$prob(\text{transmisión correcta de } c) = \left((1-p)^{31} + 31\,p\,(1-p)^{30}\right)^{2508} = 0.318\,.$$

Por último, para el Hamming $s = 6$, tendríamos que transmitir 1.11 veces más, y la probabilidad sería

$$prob(\text{transmisión correcta de } c) = \left((1-p)^{63} + 63\,p\,(1-p)^{62}\right)^{1144} = 0.117\,.$$

Podemos representar gráficamente la eficacia (en términos de la probabilidad de transmitir correctamente) de cada una de estas estrategias en función de la longitud del mensaje que se quiere transmitir:



La gráfica que que más rápidamente se va a 0 corresponde a enviar el mensaje tal como está. La gráfica que va por arriba corresponde al Hamming con parámetro $s = 2$; y las sucesivas gráficas, a valores mayores de s.

Elegir un código de Hamming con parámetro s cada vez mayor empeora su eficacia en términos de capacidad de corrección de errores; a cambio, el mensaje que hay que transmitir no se hace muy largo. La elección de un

código determinado dependerá del equilibrio que busquemos entre estas dos cuestiones. ♣

Ejercicio 5.4 Calcular la probabilidad de que una palabra de dos dígitos sea interpretada correctamente si se envía a traves de un canal que introduce errores con probabilidad p.

1. Hacerlo si la palabra se envía sin codificar;
2. Calcularla si se codifica con el código $[5, 2, 3]$ del ejercicio 4.22

Estudiar cómo dependenden de p estas probabilidades, y comparar las gráficas.

Ejercicio 5.5 Calcular la probabilidad de que los mensajes "transmitidos" en los ejercicios 4.49 y 4.50 sean descodificados sin error alguno. Comparar el resultado de este cálculo con los resultados obtenidos[61].

Ejercicio 5.6 Existe un código perfecto con parámetros $[23, 12, 7]$ (es el código de Golay binario que describimos en la sección 6.1). Repetir el análisis del ejercicio 5.4 para palabras de longitud 12 enviadas sin codificar y codificadas con el código de Golay.

Nuestro próximo ejercicio tiene como objetivo mostrar, en un modelo sencillo, una justificación estadística de la descodificación basada en el "vecino más próximo" que hemos estado empleando. Este procedimiento está basado en un criterio estadístico de *máxima verosimilitud* que podemos describir de la siguiente manera:

- observamos un fenómeno aleatorio que tiene alguna característica desconocida que deseamos determinar, por ejemplo un parámetro. En el caso de la transmisión de palabras a través de un canal la característica que buscamos conocer es la palabra que nos fue envíada.

- Como resultado de nuestras observaciones recogemos un conjunto de datos (que ya no tienen nada de azar, son una descripción de lo que ocurrió). Volviendo a la transmisión de palabras, nuestros datos son la palabra que recibimos.

- Ahora tenemos nuestros datos, y un conjunto de posibles valores para la característica desconocida de nuestro modelo (el parámetro, o la palabra enviada, dependiendo del ejemplo). Escogemos esta característica de modo que *la probabilidad de observar lo que efectivamente observamos (nuestros datos) sea máxima.*

[61] Si el ejercicio se hace en clase habrá un número de simulaciones que permitirá comparar los resultados previstos por el modelo con los resultados observados.

Esta manera de determinar una característica desconocida de un modelo es la que corresponde a seguir un criterio de máxima verosimilitud. El próximo ejercicio muestra que el sencillo modelo probabilístico que estamos empleando para describir los errores que introduce un canal binario, conduce a descodificar buscando el vecino más próximo si se adopta un criterio de máxima verosimilitud para interpretar el resultad de cada "experimento"de enviar una palabra a través de un canal con ruido.

Ejercicio 5.7 Supongamos que se transmiten palabras de un código contenido en $\mathbb{F}_2^n$ a través de un canal que introduce errores de manera independiente en cada dígito, con una probabilidad p que está en el intervalo $(0, 1/2)$. Como resultad de esta transmisión recibimos una cierta palabra $\mathbf{x}$.

1. Mostrar que si se envió la palabra $\mathbf{c}$ la probabilidad de que el canal arroje como resultado $\mathbf{x}$ es
$$p^{d(\mathbf{c},\mathbf{x})}\,(1-p)^{n-d(\mathbf{c},\mathbf{x})},$$
donde d indica la distancia de Hamming entre palabras.

2. Mostrar que la probabilidad de la parte anterior es máxima para las palabras del código que están a la menor distancia posible de $\mathbf{x}$.

6 Extensiones: hay otros mundos

Hemos visto lo más sencillo del rico mundo de la teoría de códigos, pero hay muchas otras posibilidades y aplicaciones que van más allá.

Por ejemplo, todos los códigos que presentamos están basados en el uso de dos símbolos: el 0 y el 1. ¿Podríamos usar más símbolos? Por ejemplo, ¿el conjunto de los 10 dígitos de nuestro sistema de numeración habitual? La respuesta a la primera pregunta es sí. Efectivamente podemos elaborar códigos sobre otros conjuntos de símbolos, y tales códigos existes y tienen aplicaciones. Por ejemplo, los que se emplean en los discos compactos —que describiremos someramente en la sección 6.3— están construidos sobre la base de un alfabeto de 2^8 símbolos, que resulta especialmente adecuado para trabajar con "bytes" de información. Sin embargo, la respuesta a la segunda pregunta es esencialmente negativa. La razón es que la estructura lineal que hemos explotado sistemáticamente descansa sobre el hecho de que el "alfabeto" $\{0,1\}$ admite una estructura de *cuerpo* —que hemos dado en llamar $\mathbb{F}_2$— que permite usar estos símbolos como escalares de un espacio vectorial. Éste es un ejemplo de un *cuerpo finito*, en el sentido de que sólo tiene una cantidad finita de elementos. Pero no hay cuerpos finitos con un número arbitrario p de elementos. Sólo para $p = q^k$, donde q es un número primo y k un número natural, tales cuerpos existen. El caso $k = 1$ es relativamente sencillo, porque se tiene que $p = q$ es primo, y entonces el conjunto $\{0, 1, \ldots, p-1\}$ dotado de las operaciones de suma y producto módulo p es un cuerpo. Un ejemplo de esto es $\mathbb{F}_2$, que corresponde a $p = 2$. Para valores de k mayores que 1 la construcción de estos cuerpos es de una dificultad mayor que lo que podemos abordar en estas notas, pero remitimos al lector interesado a la referencia PONER REFERENCIA. Volviendo sobre la pregunta motivó esta discusión, observemos que 10 no es de la forma p^k, por lo que no existe ningún cuerpo que tenga 10 elementos. En consecuencia, no podremos elaborar una teoría de códigos con listas de diez símbolos usando las ideas que hemos desarrollando en las secciones anteriores. Pero si podríamos hacerlo con $p = 11$, o $p = 9 = 3^2$, símbolos.

Pero incluso dentro del mundo de los códigos binarios (basados en los caracteres 0 y 1) podemos presentar algunas novedades. Un ejemplo de esto es el procedimiento de *intercalado de códigos* que discutimos en la sección 6.2, que sirve para construir a partir de dos códigos dados uno nuevo que tiene muy buenas propiedades para la corrección de algunos tipos frecuentes de error (ver la discusión de la página 92 sobre el tratamiento de los *errores en ráfaga*). El intercalado puede hacerse también sobre códigos no binarios, y de hecho se emplea de esta manera en los discos compactos.

También dentro del marco de los códigos binarios, pero con un nivel de complejidad algo mayor que lo que hemos tratado hasta ahora está el

código de Golay[62] que presentamos en la sección 6.1. Otro ejemplo que involucra matemáticas interesantes y que ha sido utilizado para la resolución de problemas tecnológicos reales.

6.1 El código de Golay binario

El **código de Golay binario** es un código[63] perfecto con parámetros $[23, 12, 7]$, al que designaremos $\mathcal{G}_{23}$.

Como su distancia mínima es 7 permite corregir hasta 3 errores. La verificación de que es perfecto es sencilla una vez que conocemos los parámetros del código. El número total de palabras de $\mathcal{G}_{23}$ es 2^{12}, y está contenido en $\mathbb{F}_2^{23}$ que tiene 2^{23} palabras. Cada bola de radio $e = 3$ en $\mathbb{F}_2^{23}$ tiene

$$1 + \binom{23}{1} + \binom{23}{2} + \binom{23}{3} = 2048 = 2^{11}$$

palabras. Obviamente

$$2^{12} \times 2^{11} = 2^{23}$$

y se satisface la igualdad en la cota de Hamming.

Pasemos ahora a describir una matriz generatriz del código $\mathcal{G}_{23}$. Consideremos el conjunto de 11 elementos $A = \{0, 1, \ldots, 10\}$ y formemos la siguiente familia de 6-subconjuntos suyos: empezamos con

$$\mathcal{F}_0 = \{0, 1, 3, 4, 5, 9\}$$

que no son sino todos los posibles cuadrados de elementos de A módulo 11. El siguiente subconjunto se obtendría del primero trasladando sus elementos una unidad a la derecha (y módulo 11):

$$\mathcal{F}_1 = \{1, 2, 4, 5, 6, 10\}$$

El siguiente, haciendo lo mismo con los elementos de $\mathcal{F}_1$, de nuevo módulo 11:

$$\mathcal{F}_2 = \{2, 3, 5, 6, 7, 0\}$$

Y así se haría hasta construir $\mathcal{F} = \{\mathcal{F}_0, \ldots, \mathcal{F}_{10}\}$. Podemos representar esta familia de subconjuntos con listas de ceros y unos, como hicimos en el

[62] El ingeniero eléctrico suizo Marcel Golay (1902–1989) es el otro pionero en este campo de los códigos correctores de errores. Trabajó en estas cuestiones casi simultáneamente que Hamming, y de hecho todavía hay quien discute la paternidad de los primeros códigos que se construyeron. En todo caso, ambos merecen un lugar destacado en esta historia.

[63] Este código fue empleado en el proyecto Voyager de la NASA, que lanzó en el año 1977 dos sondas con el objetivo de explorar los confines del sistema solar. Información sobre este programa puede encontrarse dentro de la página web de la NASA, en `vraptor.jpl.nasa.gov/voyager/voyager.html`

ejemplo 4.20; así obtendríamos la matriz 11×11 siguiente:

$$M = \begin{array}{c} \begin{array}{ccccccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \end{array} \\ \left(\begin{array}{ccccccccccc} 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \end{array} \right) \end{array} \begin{array}{c} \\ \leftarrow \mathcal{F}_0 \\ \leftarrow \mathcal{F}_1 \\ \leftarrow \mathcal{F}_2 \\ \leftarrow \mathcal{F}_3 \\ \leftarrow \mathcal{F}_4 \\ \leftarrow \mathcal{F}_5 \\ \leftarrow \mathcal{F}_6 \\ \leftarrow \mathcal{F}_7 \\ \leftarrow \mathcal{F}_8 \\ \leftarrow \mathcal{F}_9 \\ \leftarrow \mathcal{F}_{10} \end{array}$$

Que no es otra cosa que la matriz que se obtiene desplazando cíclicamente la primera fila[64].

Formemos ahora la matriz

$$G = \left(\begin{array}{ccc|c|ccc} & & & 0 & & & \\ & I_{11} & & \vdots & & M & \\ & & & 0 & & & \\ \hline 0 & \cdots & 0 & 1 & 1 & \cdots & 1 \end{array} \right)$$

donde I_{11} es la matriz identidad 11×11. Las 12 filas de esta matriz forman un conjunto linealmente independiente en $\mathbb{F}_2^{23}$: el código $\mathcal{G}_{23}$ es el subespacio lineal de $\mathbb{F}_2^{23}$ generado por las filas de G (su dimensión es 12).

Ahora que hemos descrito el código de Golay, merece la pena comentar que los únicos códigos binarios lineales perfectos que existen son:

- Los códigos de repetición, con $n = 2e + 1$ (impar), para los que ya hemos verificado esta propiedad en el ejercicio 4.13, página 47. Son códigos $[n, 1, n]$-lineales, y corrigen e errores.
- Los s-códigos de Haming (también lineales), de parámetros $[2^s - 1, 2^s - s - 1, 3]$: corrigen un error.
- El código binario de Golay, $\mathcal{G}_{23}$, lineal de parámetros $[23, 12, 7]$, que corrige tres errores.

[64] Esta familia de 6-subconjuntos cumple una serie de propiedades interesantes: cada par de elementos de A están en exactamente 3 de los subconjuntos de la familia $\mathcal{F}$ (basta contar en la matriz en cuántas filas aparece cada pareja de unos, hay apenas 55 parejas de unos). Además, cada elemento de A está en 6 de los subconjuntos de la familia (hay seis unos por columna).

Habría que añadir los códigos tales que $\mathcal{C} = \{0,1\}^n$, que son perfectos... pero no detectan ni corrigen error alguno, así que no son interesantes.

6.2 Códigos que colaboran: intercalado

El objetivo de esta sección es presentar las nociones de **intercalado** y **código producto**, que permiten fabricar a partir de dos códigos lineales $\mathcal{C}_1$ y $\mathcal{C}_2$ un nuevo código $\mathcal{C}$ en el que $\mathcal{C}_1$ y $\mathcal{C}_2$ cooperan para la corrección de errores. Esta construcción se describe en el ejercicio 6.2. Los códigos que se emplean en los discos compactos son de este tipo. Los describiremos con cierto detalle en la sección 6.3, y mostraremos como el intercalado de dos códigos mejora su habilidad para corregir.

Hasta el momento hemos tratado con códigos que son subespacios lineales de $\mathbb{F}_2^n$, que está formado por listas de ceros y unos. Los códigos producto serán subespacios lineales del conjunto de las *matrices* de ceros y unos[65]. Nuestro próximo paso será verificar que las matrices admiten una estructura lineal muy natural. Llamaremos $M^{m\times n}(\mathbb{F}_2)$ al conjunto formado por la matrices $m \times n$ con entradas en $\mathbb{F}_{\not=}$.

Ejercicio 6.1 Mostrar que $M^{m\times n}(\mathbb{F}_2)$ forma un espacio vectorial sobre $\mathbb{F}_2$ si las operaciones de suma entre matrices y producto por un escalar se definen de la manera habitual.

Ahora supongamos que tenemos dos códigos lineales $\mathcal{C}_1$ y $\mathcal{C}_2$. Sus parámetros serán $[n_1, k_1, d_1]$ y $[n_2, k_2, d_2]$ respectivamente.

Ejercicio 6.2 INTERCALADO Y CÓDIGOS PRODUCTO

1. Mostrar que las matrices de $M^{n_2\times n_1}(\mathbb{F}_2)$ tales que todas sus filas están en $\mathcal{C}_1$ y todas sus columnas en $\mathcal{C}_2$ forman un subespacio lineal de $M^{n_2\times n_1}(\mathbb{F}_2)$. Este subespcio será nuestro nuevo código, al que llamaremos $\mathcal{C}$

2. Sean G_1 y G_2 matrices generatrices de los códigos $\mathcal{C}_1$ y $\mathcal{C}_2$. Mostrar que la aplicación[66] f que a cada matriz A de ceros y unos de dimensión $k_2 \times k_1$ le asocia la matriz $\tilde{A}$, de dimensión $n_2 \times n_1$, definida por

$$\tilde{A} = G_2^{\mathrm{t}} A G_1$$

es una codificación de las palabras[67] de $M^{k_2\times k_1}(\mathbb{F}_2)$, en el sentido de que es una aplicación lineal y biyectiva entre este conjunto y el código $\mathcal{C}$.

[65] Las listas de ceros y unos son un caso particular de matrices con una fila o una columna, según como se las escriba. Por otra parte, una matriz de tamaño $m \times n$ puede pensarse como una lista de longitud mn (en cualquiera de los dos casos hay que dar mn números). Sólo se trata de una manera diferente de ordenar la información.

[66] En un lenguaje más formal escribiríamos

$$\begin{array}{rcl} f : M^{k_2\times k_1}(\mathbb{F}_2) & \to & \mathcal{C} \\ A & \mapsto & \tilde{A} = G_2^{\mathrm{t}} A G_1 \end{array}$$

[67] Efectivamente, en este ejemplo las palabras son matrices.

3. Probar que si las matrices G_1 y G_2 están en forma estándar la codificación de A construye $\widetilde{A}$ de la siguiente forma: sitúa la matriz inicial A en la esquina superior izquierda de la nueva, completa las filas hasta longitud n_1 con los dígitos de control que hacen que cada fila sea una una palabra de $\mathcal{C}_1$, y luego las columnas para que sean palabras de $\mathcal{C}_2$. Mostrar que $\tilde{A}$ también puede obtenerse completando primero columnas y después filas, y que ambos procedimientos arrojan el mismo resultado.

4. Demostrar[68] que la distancia mínima del código $\mathcal{C}$ es $d_{\mathcal{C}} = d_1\, d_2$. Concluir que $\mathcal{C}$ es un $[n_1 n_2, k_1 k_2, d_1 d_2]$ código lineal.

5. Calcular el código que resulta de intercalar el código lineal $[5, 2, 3]$ del ejercicio 4.22 consigo mismo.

En la página 92 discutimos como el intercalado de códigos mejora su habilidad para corregir errores. Hemos incluido allí un ejercicio en el que sugerimos al lector interesado experimentar con el código de la parte 5 del ejercicio anterior para apreciar las posibilidades que ofrecen los códigos producto.

6.3 Descripción de los códigos empleados en los discos compactos

Un disco compacto es, esencialmente, una pista (de unos 5 kilómetros de longitud) en forma de espiral que consta de una sucesión de "valles" y "llanuras" y que se utiliza para almacenar información digital. La pista es escaneada por un rayo láser que interpreta las transiciones entre valles y llanuras (o al revés) como unos, mientras que el resto de las veces lee ceros[69]. Cada *bit* de información (*bit* de canal es el nombre técnico) corresponde a una longitud de la pista de 0.3 μm. En la figura 2 aparece un fragmento de la pista (tres valles), con la vista desde arriba, en sección y la secuencia de ceros y unos almacenada (las líneas discontinuas marcan los intervalos de 0.3 μm correspondientes a cada *bit*).

Queremos almacenar una señal analógica (música, por ejemplo) en un Compact Disc: el primer paso consiste en digitalizarla. Para ello se muestrea la señal, tomando medidas 44100 veces por segundo. Cada toma de datos es un número entre 1 y $2^{16} - 1$, esto es, una lista 16 posiciones con ceros y unos (al pasar a base binaria). En realidad, en cada ocasión (en el sistema estereofónico) se toman dos medidas, de manera que obtenemos una lista de 32 posiciones. Esta lista se suele entender como una sucesión de 4 *bytes* (listas de 8 posiciones) sucesivos; el alfabeto considerado no está formado

[68] En la sección 6.3 este cálculo está hecho en un caso particular.

[69] Existen algunas restricciones sobre las secuencias de ceros y unos que se pueden almacenar; por ejemplo, cada valle o llanura ha de comprender tres *bits*, para evitar que la luz encuentre dos transiciones consecutivas. También ha de ocurrir que dos unos no pueden estar separados por más de 10 ceros. No entraremos en los procedimientos que permiten adaptar una sucesión arbitraria de ceros y unos a esas restricciones.

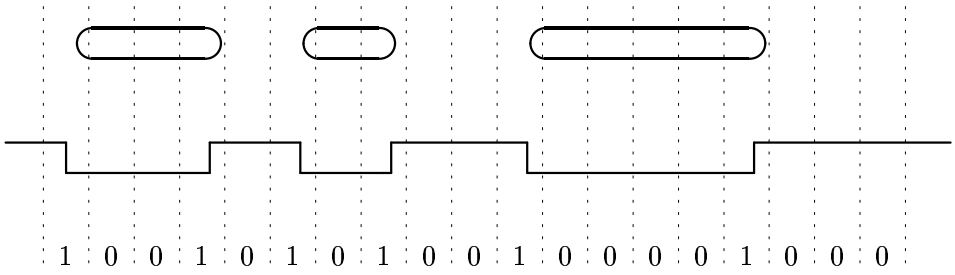


Figura 2: Un fragmento de pista de un disco compacto

por *bits* (ceros o unos), sino por *bytes*. El conjunto de todos los *bytes* posibles tiene 2^8 elementos, por lo que el cuerpo en el que se realizarán las operaciones es el cuerpo finito $\mathbb{F}_{2^8}$ que tiene 2^8 elementos. Y cada medida da lugar a un elemento de $\mathbb{F}_{2^8}^4$.

Pero estos *bytes* de información no se almacenan directamente en el CD. Como veremos luego, cada segundo de música se corresponde con varios millones de *bits* marcados en la pista. Aunque el lector tuviera una proporción de errores muy baja, del orden de 10^{-4}, aún ocurrirían muchísimos errores por segundo. Conviene, pues, utilizar una codificación que permita corregir errores. Lo habitual es tomar cada sucesión de 24 *bytes* y codificarla como una palabra de 32 *bytes* (luego veremos cómo). Tampoco son estas palabras las que se almacenan en la pista del CD. Primero, a cada palabra se le añade un *byte* que marca la posición de la pista en la se graba. Cada uno de estos 33 *bytes* se transforman, a la hora de grabar[70] en 33 grupos de 17 *bits* de canal (en lugar de los 8 originales). Entre cada uno de estos grupos se intercala una secuencia de 27 dígitos de sincronización (para indicar dónde empieza la siguiente palabra codificada). No entraremos en detalles sobre cómo se efectúa todo este proceso; pero podemos hacer el cálculo del número de *bits* de canal necesarios para almacenar un segundo de música. En este periodo de tiempo se realizan 44100 tomas de datos, que se traducen en 176.400 *bytes*. Éstos se agrupan en 7350 grupos de 24 *bytes*, que se codifican como 7350 palabras de 32 *bytes*. De ahí se pasa a 7350 grupos de 33 *bytes*. Pasar a los *bits* del canal requiere, finalmente,

$$7350\ (33 \times 17 + 27) = 4,321.800\, \textit{bits}\ \text{del canal.}$$

Como se ve, una cantidad enorme.

[70]Para cumplir las restricciones de las que hablábamos en la nota al pie anterior sobre cómo almacenar dígitos en la pista.

Pero vayamos con el proceso de codificación. Como hemos comentado, en el CD se utiliza $\mathbb{F}_{2^8}$ como alfabeto (se trabaja con *bytes*), y se usan dos códigos $\mathcal{C}_1$ y $\mathcal{C}_2$, sobre este alfabeto. Los parámetros de $\mathcal{C}_1$ son $[28, 24, 5]$, lo que significa que sus palabras están formadas por 28 elementos de $\mathbb{F}_{2^8}$, y permite codificar palabras de longitud 24. La distancia mínima es 5 y corrige, en consecuencia, hasta dos errores. El código $\mathcal{C}_2$ tiene parámetros $[32, 28, 5]$. Codifica palabras de longitud 28 en palabras de 32 caracteres en $\mathbb{F}_{2^8}$, y también corrige dos errores[71]. Vamos a mostrar ahora como dos códigos con estos parámetros pueden colaborar eficientemente en la corrección de errores, por medio de un proceso de *intercalado* que produce un *código producto*, tal como vimos en el ejercicio 6.2.

Supongamos que las matrices generatrices de ambos códigos están dadas en forma estándar, de manera que la codificación con cualquiera de ellos supone añadir al final 4 caracteres de control[72]. La codificación de una matriz A de dimensiones 24×28 cuyas entradas por medio del intercalado de los códigos $\mathcal{C}_1$ y $\mathcal{C}_2$ produce una matriz $\widetilde{A}$ de dimensiones 28×32 de la siguiente forma: coloca la matriz inicial en la esquina superior izquierda de la nueva, y completa la matriz exigiendo que cada fila sea una palabra de $\mathcal{C}_2$ y cada columna, una de $\mathcal{C}_1$. Esto es, cada fila original de 28 posiciones se amplía[73] a una de 32 añadiéndole los 4 caracteres de control que exija la pertenencia a $\mathcal{C}_2$; y a cada una de las 32 columnas de 24 posiciones que resultan de hacer esta operación le añadimos los 4 caracteres necesarios para que estemos en $\mathcal{C}_1$.

$$\widetilde{A} = \left(\begin{array}{cc} \multicolumn{1}{c}{} & \multicolumn{1}{c}{4} \\ \boxed{} & \boxed{} \\ 4\ \boxed{} & \boxed{} \end{array} \right)$$

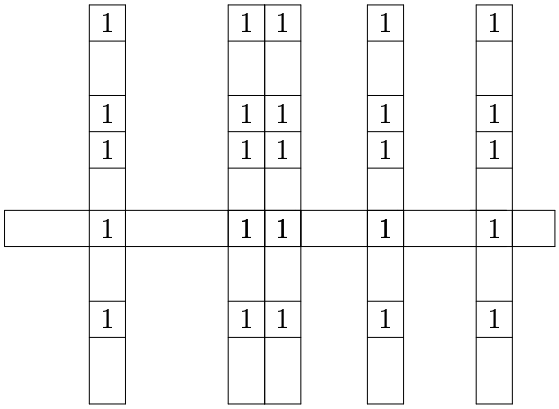
Por cada elección de A tendremos una $\widetilde{A}$. Pues bien, estas matrices son las palabras de nuestro código producto. Este código nuevo, cuyas palabras constan de 28×32 caracteres, tiene distancia mínima 25. Vamos a verificarlo. Como el código es lineal, bastará calcular el peso mínimo de entre todas las

[71] Los códigos con estos parámetros que se emplean en los discos compactos reciben el nombre de **códigos de Reed-Solomon**. Su descripción excede el propósito de estas páginas, pero puede encontrarse, por ejemplo, en PONER REFERENCIA

[72] Ya hemos mencionado que los caracteres para estos códigos no se reducen a ceros y unos, pero no producirá mayor inconveniente al lector imaginar a partir de este momento que seguimos trabajando sólo con los caracteres 0 y 1 que hemos empleado en los códigos binarios.

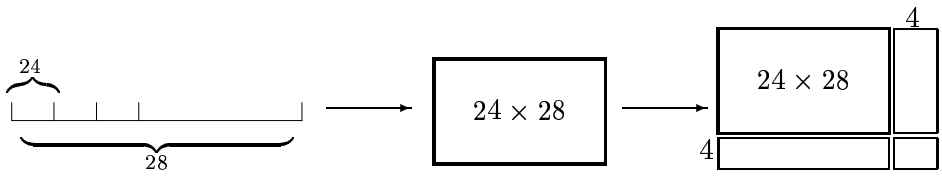
[73] Ya hemos observado que da lo mismo hacer esto completando primero las filas y luego las columnas, o al revés (ver el ejercicio 6.2)

matrices que podemos formar (excepto la matriz nula). Esto es, calcular cuál es el mínimo número de caracteres no nulos que puede tener una matriz de éstas. Recordemos que tanto las palabras de $\mathcal{C}_1$ como las de $\mathcal{C}_2$ tenían al menos 5 entradas distintas de cero (excepto los $\mathbf{0}$ de cada caso, claro). La matriz más barata, en términos de la cantidad de coeficientes no nulos en ella, que podemos formar se obtendría de la siguiente manera: tomamos la palabra de $\mathcal{C}_2$ que sólo contiene 5 entradas no nulas (sabemos que existe). En las cinco posiciones en que éstas estén colocamos, como columna, la palabra de $\mathcal{C}_1$ que también tenga sólo 5 caracteres distintos de cero. Esto puede hacerse, por ejemplo, como en el siguiente diagrama (en el que hemos marcado con un 1 las entradas no nulas, y luego hemos fabricado un arreglo en el que coincidan un 1 de la columna con cada 1 de la fila):



Esta matriz es una palabra del código y tiene peso 25. Así que este código producto corrige hasta 12 errores.

Veamos cómo se aplica esta codificación al caso del CD. Dada la secuencia de bytes, consideramos palabras de longitud 24 ; cada 28 de estas palabras (al ponerlas como columnas) forman la matriz con la que construimos la palabra del código (con el procedimiento expuesto antes):



Supongamos entonces que hemos codificado nuestra sucesión original en palabras de éstas. Vayamos ahora con la descodificación. En principio, el código corrige hasta 12 errores en la matriz, pero aún puede ser mejor, si lo utilizamos astutamente. Por ejemplo, supongamos que recibimos una matriz en la que se han producido 22 errores: por ejemplo, hay 13 columnas con un error, una con 2, una con 3 y una con 4. En total hay más de 12 errores,

así que parece que no hay nada que hacer. Pero hagámoslo de la siguiente manera: utilicemos el código $\mathcal{C}_1$ para revisar las columnas, pero con la restricción de que sólo se corrijan aquéllas en las que haya un único error. En estas condiciones, corregirá las 13 columnas con 1 error y detectará errores en las columnas con 2 y 3 errores (recordemos que la distancia mínima en $\mathcal{C}_1$ es 5). La columna con 4 errores quizás esté a distancia 1 de otra palabra de $\mathcal{C}_1$, así que en esta aplicación la llevaríamos a esa palabra, añadiendo un error. Pongámonos en este peor caso. Tras esta pasada correctora, tenemos 29 columnas correctas, 2 con errores (2 y 3, respectivamente) y otra en la que se han cometido 5 errores. Ahora decidimos borrar las dos columnas "sospechosas", y con esta nueva matriz pasamos a aplicar la función correctora del código $\mathcal{C}_2$, apropiado para las filas. ¿Qué se encuentra este código? Algunas filas tendrán dos borrones, otras estarán correctas y 5 de ellas tendrán dos borrones y un error. Pero un código con distancia mínima 5, como es el caso de $\mathcal{C}_2$, es capaz de corregir tales situaciones (ver el ejercicio 4.8).

Justifiquemos, por último, por qué este método de codificación es útil para **errores en ráfaga**[74]. Supongamos que transmitimos cada palabra (matriz) del código por filas. Y que se produce un error que hace que toda la primera fila se transmita mal. Visto por filas, es un error grave (hay 32 posiciones cambiadas). Pero visto por columnas, el error se distribuye entre las 32 columnas; y corregir un error por columna es tarea sencilla para $\mathcal{C}_1$. Este método del código producto es un ejemplo de intercalado: enviamos las listas por filas, pero las leemos por columnas. Así los errores consecutivos se dispersan y es posible corregirlos.

Ejercicio 6.3 Experimentar con el código del ejercicio 6.2, parte 5 para ver que el proceso de intercalado mejora la habilidad de corregir ciertos errores por encima de la que su distancia mínima prediría. Sugerencia: tener en cuenta la discusión anterior.

Hemos visto entonces una somera descripción de los códigos que permiten corregir errores en la lectura de un disco compacto. La codificación que se usa transforma matrices de tamaño 24×28 en matrices 28×32. La tasa de transmisión de estos códigos es entonces

$$\frac{24 \times 28}{28 \times 32} = 0.75,$$

por lo que podemos decir que un disco compacto contiene un 75% de música y un 25% de matemáticas.

[74]Errores que afectan una porción del mensaje enviado. Es decir, a varios dígitos consecutivos. Subrayemos que este tipo de errores, que realmente ocurren en la práctica, no pueden ser analizados con un modelo que suponga que los errores en lugares distintos del mensaje son independientes, como los que discutimos en la sección 5. Vemos entonces que esta situación real, que interesa comprender, requiere para su análisis de unas matemáticas más elaboradas que las que hemos visto en este texto. Algo que encontramos ¿por qué ocultarlo? realmente estupendo.

A Tabla de caracteres según la norma ISO/IEC 8859-1 (Latin 1)

Presentamos aquí una tabla con la codificación de carácteres que corresponde a la norma ISO/IEC 8859-1. En primer lugar observemos que la tabla está escrita en un sistema de numeración en base 16, lo que hace más cómoda su representación y la traducción al lenguaje binario. Como hacen falta 16 "dígitos" usamos letras para los que son más grandes que 9, según el esquema

$$\text{A} = 10, \quad \text{B} = 11, \quad \text{C} = 12, \quad \text{D} = 13, \quad \text{E} = 15.$$

	2	**3**	**4**	**5**	**6**	**7**	**A**	**B**	**C**	**D**	**E**	**F**
0		0	@	P	`	p		°	À	??	à	??
1	!	1	A	Q	a	q	¡	±	Á	Ñ	á	ñ
2	”	2	B	R	b	r	??	^{2}	Â	Ò	â	ò
3	#	3	C	S	c	s	£	^{3}	Ã	Ó	ã	ó
4	$	4	D	T	d	t	??	´	Ä	Ô	ä	ô
5	%	5	E	U	e	u	??	μ	Å	Õ	å	õ
6	&	6	F	V	f	v	??	¶	Æ	Ö	æ	ö
7	’	7	G	W	g	w	§	•	Ç	×	ç	÷
8	(	8	H	X	h	x	¨	¸	È	Ø	è	ø
9	)	9	I	Y	i	y	©	^{1}	É	Ù	é	ù
A	*	:	J	Z	j	z	^{a}	^{o}	Ê	Ú	ê	ú
B	+	;	K	[	k	{	«	»	Ë	Û	ë	û
C	,	<	L	\	l	—	¬	$\frac{1}{4}$	Ì	Ü	ì	ü
D	-	=	M	]	m	}		$\frac{1}{2}$	Í	Ý	í	ý
E	.	>	N	^	n	~	®	$\frac{3}{4}$	Î	??	î	??
F	/	?	O	_	o		¯	¿	Ï	ß	ï	ÿ

La tabla debe leerse de la siguiente manera: cuando nos desplazamos dentro de una misma columna varían las unidades, en tanto que al movernos de una columna a otra por una misma fila vamos variando la "cifra de las decenas" (en realidad es la cifra de las "unidades de 16", y la diferencia entre dos lugares de la misma fila en columnas consecutivas es 16). Por ejemplo, a la letra P le corresponde el número 50 (en hexadecimal), y a la Ñ el D1. Por supuesto, podemos volver al sistema decimal. Por ejemplo

$$\begin{aligned} \text{P} &\mapsto 50 = 5 \times 16 + 0 = 80, \\ \tilde{\text{N}} &\mapsto \text{D1} = \text{D} \times 16 + 1 = 241. \end{aligned}$$

Nos hemos limitado a reproducir la parte de la tabla que corresponde a la codificación de caracteres gráficos, por eso faltan las columnas correspondientes a los números entre 0 y 1F, y entre 80 y 9F[75]. Estas columnas se reservan para la codificación de funciones de control (que afectan el procesamiento, almacenamiento, transmisión e interpretación de los datos).

Los lugares que corresponden al 20 y A0 (32 y 160 respectivamente en notación decimal) corresponden a espacios, por lo que aparecen en blanco en la tabla.

Ejercicio A.1 Generar una nueva tabla con la codificación binaria de los caracteres.

[75]Nos hemos expresado en lenguaje hexadecimal, en la notación decimal habitual las columnas que faltan corresponden a los números que van del 0 al 31, y del 1 28 al 159.