Universidad de El Salvador

Facultad de Ciencias Naturales y Matemática

Escuela de Matemática

"Estudio de los códigos perfectos y nociones similares"

Tesis para obtener el título de:
Licenciado en Matemática

Presentado por:
Br. Luisantos Bonilla Mejía, BM12028

Asesor interno:
M.Sc. Ingrid Carolina Martínez Barahona
Universidad de El Salvador
Asesor externo:
Doctor Sergio Roberto López-Permouth
Ohio University

Ciudad Universitaria, 17 de septiembre del 2018

Autoridades

Universidad de El Salvador

Rector:

MSc. Roger Armando Arias Alvarado

Vice-Rector Administrativo:

Ing. Nelson Bernabé Granados

Secretario General:

Lic. Cristobal Hernán Ríos Benítez

Fiscal General:

Lic. Rafael Humberto Peña Marín

Facultad de Ciencias Naturales y Matemática

Decano:

Lic. Mauricio Hernan Lovo Córdova

Vice-Decano:

Lic. Carlos Antonio Quintanilla Aparicio

Secretaria:

Licda. Damaris Melany Herrera Turcios

Escuela de Matemática

Director:

Dr. José Nerys Funes Torres

Secretaria:

MSc. Alba Idalia Córdova Cuéllar

Agradecimientos

Principalmente a Dios por haberme permitido terminar una etapa muy importante de mi vida como es mi carrera y sobre todo culminar este proyecto que marca un antes y un después de mi vida tanto personal como académica.

A mi madre por haberme apoyado todo este tiempo permitiéndome estudiar y terminar una carrera universitaria a pesar de todas las dificultades que hemos vivido como familia y por esos ánimos y consejos que me dió en los momentos más importantes de mi vida como estudiante. A mi padre que aunque no esté en vida sé que está muy alegre porque he logrado culminar mis estudios universitarios y así permitirme a mí y a mi madre una vida mejor y sobre todo aquellos consejos que de pequeño me han guiado en la clase de persona que soy. A mis hermanos que de una u otra forma me han estado apoyando incondicionalmente en todo este gran proceso que ha sido mi estudio. A mi novia Patricia Rodríguez por su apoyo incondicional y consejos que me han permitido ser mejor persona cada día.

A mis asesores de trabajo: a la MSc. Ingrid Martínez por el tiempo que ha dedicado a la organización de este documento, sus correcciones, consejos y enseñanzas en el rigor matemático usado en el documento, al Dr. Sergio López-Permouth por su apoyo, su dedicación y tiempo en cada etapa de mi investigación, en las sugerencias y guía para lograr un mejor entendimiento de los contenidos y sobre todo por haber creído en mi para poder desarrollar este proyecto. Al jurado calificador MSc. Yoceman Adony Sifontes, Dr. Riquelmi Cardona, por el tiempo tomado para la revisión y corrección de mi trabajo.

Índice general

Resumen

Los códigos perfectos son una estructura matemática de la teoría de código, la cual es una rama del álgebra que trata sobre la detección y corrección de errores en la transmisión de información mediante mecanismos matemáticos. Por ejemplo, si nosotros queremos enviar un mensaje de un país a otro, nuestro mensaje lo codificamos mediante ceros y unos y mandamos esa cadena de símbolos; desgraciadamente dicha cadena no llega igual, o sea no es la misma cadena de símbolos que se mandó originalmente y la pregunta normal que surge es ¿cómo recuperar la cadena enviada a partir de la recibida? Pues de eso se trata la teoría de código ¿cómo podemos detectar y corregir los errores ocurridos en la transmisión de información?

En el presente trabajo nos ocuparemos de un tipo específico de código (códigos perfectos) los cuales permiten no solo poder detectar sino también corregir hasta un cierto número de errores a la vez.

Introducción

El presente trabajo está dividido en tres capítulos. El capítulo 1 es una introducción a la teoría de código en general, exponiendo las nociones básicas de la detección y corrección de errores, así preparar el universo de trabajo el cual es un espacio vectorial, definiéndole apropiadamente su suma y producto por escalar, así como una norma y métrica para el espacio. En el capítulo 2 nos dedicamos a estudiar las propiedades de un tipo especifico de código, los cuales son *códigos lineales*, este estudio nos llevará a plantearnos muchos resultados como es el caso de una matriz generadora para el código y el complemento ortogonal del código (dual del código), y al descubrir estas matrices podremos definir la matriz de control de paridad para el código; dicha matriz será de gran utilidad ya que nos da información vital del código, ella es capaz de decirnos la dimensión del código lineal y la distancia del código. Finalmente, con la matriz de control de paridad lograremos describir un mecanismo que nos dirá si una secuencia de ceros y unos está dañada o no, y a la vez decirnos donde está dañada.

En el capítulo 3 estudiaremos ciertas restricciones para el tamaño del código en función de la cantidad de errores que queramos detectar y corregir, esto nos llevará a definir los códigos perfectos que permiten de alguna manera cubrir mediante "bolas" todo el espacio vectorial y así poder lograr corregir correctamente una cantidad finita de errores a la vez. Veremos los diferentes tipos de códigos perfectos que existen y sus habilidades para la detección y corrección de errores, además de poder crear un algoritmo eficiente (mejor que el presentado en el capítulo 2) para la corrección de errores. Además, en el apéndice A ilustramos una implementación de un tipo de código perfecto en el software matemático *Octave*.

10

Metodología

Para realizar este estudio, haremos uso de la siguiente estructura:

1. Revisión bibliográfica; se hará uso de diferentes bibliografías, de distintos autores, para llegar a desarrollar los contenidos, sin embargo se tendrá un libro base el cual es: **Coding Theory and Cryptography** de D.R. Hankerson, D.G. Hoffman.
2. Demostración de los teoremas importantes; se entenderán y desarrollaran las demostraciones de los teoremas más importantes de los códigos perfectos, dando una demostración clara y detallada.
3. Implementación de los resultados obtenidos mediante algoritmos.
4. Los contenidos a desarrollar son:
   - 4.1 Introducción a la teoría de código.
   - 4.2 Códigos lineales.
   - 4.3 Códigos perfectos.

Capítulo 1

Introducción a la teoría de código

En nuestro mundo tecnificado la comunicación es la clave de ese desarrollo tecnológico, desde una simple llamada entre familiares hasta hacer vídeo conferencias desde la comodidad del hogar a una Universidad, todo esto requiere una serie de transmisión de datos que por nuestra mala suerte nunca llega de la misma manera en que la mandamos. Por ese hecho de que nuestros datos son afectados es necesario desarrollar herramientas que permitan una recuperación de nuestra información y de eso se trata este capítulo de dar las bases de la detección y corrección de errores para un código específico. En este arduo camino definiremos herramientas matemáticas que serán de gran ayuda para alcanzar nuestras metas como lo es definir una distancia entre líneas de código y una norma para estas líneas de código (llamadas palabras más adelante). Finalmente descubriremos que nuestros códigos viven en un espacio vectorial con propiedades interesantes y que más adelante veremos que lo hace especial como estructura matemática.

1.1. Conceptos básicos

Para iniciar nuestro estudio comenzaremos dando los conceptos básicos necesarios para el desarrollo de la teoría de código.

Definición 1.1.1. *Se llamará **alfabeto** a un conjunto K finito, que está compuesto por símbolos, lo que significa:*

$$K = \{x_1, x_2, ..., x_n\}.$$

En este trabajo utilizaremos el alfabeto que solo contiene dos elementos, y dichos elementos serán 0 y 1. Conoceremos a este alfabeto como **alfabeto binario**, y es:

$$K = \{0, 1\}.$$

Los elementos del alfabeto binario, tienen un nombre y este se conocen como dígitos, así:

Definición 1.1.2. *Llamaremos a un* 0 *y a un* 1 *un* ***dígito****.*

Definición 1.1.3. *Una* ***palabra*** *es una secuencia de dígitos.*

Ejemplo 1.1.4. *Algunos ejemplos de palabras son:*

- 10101111, 111111, 1001 y 00000001.

Definición 1.1.5. *El* ***tamaño*** *de una palabra es el número de dígitos que tiene.*

De los ejemplos anteriores de palabras podemos calcular su tamaño de acuerdo a la definición dada, el tamaño de cada una de las palabras del ejemplo anterior son $8, 6, 4$ y 8, respectivamente.

A modo de comodidad, para uso de la definición de tamaño de una palabra, denotaremos $\ell(x)$ como la función que nos devuelve el tamaño de la palabra (**función tamaño**), así para los ejemplos anteriores tendremos:

Ejemplo 1.1.6. *Usando las palabras del ejemplo 1.1.4.*

a) $\ell(10101111) = 8$

b) $\ell(111111) = 6$

c) $\ell(1001) = 4$

d) $\ell(00000001) = 8$

Definición 1.1.7. *Entenderemos como* ***canal*** *al medio por el cual transmitimos las palabras.*

Definición 1.1.8. *El* ***ruido*** *es el fenómeno que sucede en el canal y cambia uno o varios dígitos de las palabras; este cambio hace que un* 1 *se vuelva un* 0 *y viceversa.*

Definición 1.1.9. *Llamaremos* ***canal binario*** *al medio por el cual es enviada o transmitida la palabra.*

El hecho que se agrega la palabra **binario** es porque solo se usan dos dígitos, en nuestro caso cero y uno. Cada dígito es transmitido de forma mecánica, eléctricamente, magnéticamente o por cualquier otro medio que permita la transmisión de dos pulsaciones.

Ejemplo 1.1.10. *Ejemplos de canales binarios:*

1. La atmósfera.
2. El espacio.
3. Cables de cobre.

Definición 1.1.11. *Un* ***código binario*** *es un conjunto* **C** *de palabras.*

Ejemplo 1.1.12. *El código que consiste de todas las palabras de longitud dos es:*

$$\mathbf{C} = \{00, 10, 01, 11\}$$

Definición 1.1.13. *Un **código de bloque** es un código que tiene todas sus palabras de igual tamaño.*

En el desarrollo de la teoría solo se consideran los códigos de bloque, de esta forma para nosotros **código** siempre significara **códigos de bloque binario**.

Definición 1.1.14. *Sea* $\mathbf{C}$ *un código, llamaremos **código palabra** a todas las palabras que pertenecen a* $\mathbf{C}$ *y* $|\mathbf{C}|$ *significa el número de códigos palabras en* $\mathbf{C}$*.*

Además de las definiciones anteriores serán necesarias hacer ciertas suposiciones sobre el canal. Estas suposiciones serán necesarias para la formación de la teoría que se formulará.

Las suposiciones que se harán son las siguientes:

1. Un código palabra de longitud n consiste de 0 y 1 y es recibida como una palabra de longitud n que consiste en 0 y 1, a pesar de que no necesariamente sea la misma palabra que fue enviada.

2. En el canal no es difícil identificar la primera palabra transmitida. Así, si usamos códigos palabras de longitud 3 y se recibe 101001111, sabemos que las palabras recibidas en orden son, $101, 001$ y 111.

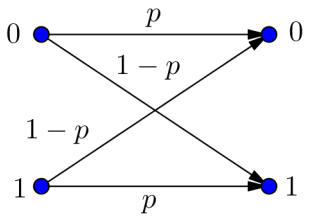
3. El ruido afecta aleatoriamente a cada dígito, esto quiere decir que no afecta a un bloque sino a cada dígito individualmente, y además cada dígito tiene la misma probabilidad de ser afectado por el ruido.

Es necesario indicar el tipo de canal que estaremos utilizando el cual es **canal binario simétrico**, y se define de la siguiente manera.

Definición 1.1.15. *Un **canal binario** es **simétrico**, si* 0 *y* 1 *son transmitidos con igual precisión; es decir la probabilidad de recibir el dígito correcto es independiente de cual dígito,* 0 *o* 1 *se está transmitiendo.*

La **confiabilidad** de un canal binario simétrico (**BSC**; por sus siglas en inglés *binary symmetric channel*) es un número real p, tal que $0 \leq p \leq 1$, donde p es la probabilidad que un dígito enviado es el dígito recibido.

Como p denota la probabilidad de que un dígito enviado sea el dígito que se recibe, entonces $1 - p$ representa la probabilidad de que un dígito recibido no sea el dígito enviado. El siguiente diagrama muestra como un **BSC** opera:



Por último de acuerdo a lo definido anteriormente podemos concluir:

1. Si $p = 1$, entonces no existe cambios o alteraciones en la transmisión.
2. Si $p = 0$, entonces indicará que todos los dígitos cambian y entonces basta con cambiar nuevamente los dígitos una vez recibido para saber la información que realmente se mandó; esto lo hace equivalente a un canal perfecto.
3. Si $p = 1/2$, en este caso no se puede **asegurar** nada pues tanto la probabilidad que un dígito cambie es la misma que un dígito no cambie.
4. Un canal es más confiable que otro si la *confiabilidad* es mayor.

Algo que es importante mencionar es que un canal con probabilidad $0 < p \leq 1/2$ puede ser convertido fácilmente a un canal con probabilidad $1/2 \leq p < 1$. Por último como en los casos $p = 0, p = 1/2$ y $p = 1$, son casos poco productivos, desde ahora en adelante trabajaremos con un canal **BSC** con probabilidad p, donde $1/2 < p < 1$.

1.2. ¿Existe una manera para saber si una palabra de código sufrió un error?

Es necesario saber si existe alguna forma de conocer o detectar que la palabra que recibimos es o no la palabra que se envió, ya que de no existir alguna manera para alcanzar este objetivo, no tendría sentido estudiar este fenómeno.

La idea principal de cómo detectar un error, es que si recibimos una palabra y esta palabra no está en nuestro código (o sea que no es una palabra de código), entonces sabemos que ha ocurrido un error en la transmisión del mensaje. Por otro lado, si la palabra que recibimos está en el código no podemos precisar si dicha palabra sufrió un cambio en sus dígitos, en este caso no seremos capaces de detectar un error.

De hecho la idea de corregir es más complicada de lo que se describió antes, por ejemplo, en los celulares de hoy en día (de hecho cualquier dispositivo electrónico), cuando se comienza a escribir una palabra, éste le despliega un serie de palabras como sugerencias; por ejemplo, si se escribe *ca*, el programa integrado en el dispositivo puede pensar que se quiere escribir *va*, *caso* o *carta*, en este sentido lo que hace el programa es detectar cuales

son las palabras más probables de ser escritas, en un número menor de cambios de letras, a partir de las primeras ya escritas.

Para consolidar dichas ideas, estudiemos algunos casos particulares de ciertos tipos de códigos, también debemos tener presente que no se crean ni se destruyen dígitos en el momento de la transmisión.

Ejemplo 1.2.1. *Sea el código $C_1 = \{00, 01, 10, 11\}$ y consideremos como universo todas las palabras de tamaño 2, así el código C_1 es incapaz de detectar y corregir errores, pues cada palabra recibida, será un código palabra.*

Ejemplo 1.2.2. *Modifiquemos el código C_1, cambiando cada palabra de código, por la repetición de cada palabra tres veces, así tendremos:*

$$C_2 = \{000000, 010101, 101010, 111111\}.$$

*Este código recibe el nombre de **código de repetición**. En este caso si recibimos la palabra 111010, esta palabra no se encuentra en el código C_2, esto quiere decir que ha ocurrido un error, y de hecho el código 101010, con solo un cambio puede formar la palabra recibida, y las demás palabras requieren un mayor número de cambios en sus dígitos, de aquí se puede concluir que la palabra recibida se puede corregir a la palabra de código 101010. A la palabra con menores cambios posibles es conocida como palabra más cercana a la palabra recibida, dicha idea se estudiará más adelante de forma más detallada.*

Ejemplo 1.2.3. *Ahora modifiquemos C_1, agregando a cada palabra de código un dígito al final, de la manera siguiente, cero si hay un número par de 1's y 1 si hay un número impar de 1's. De esta manera tendremos el siguiente código;*

$$C_3 = \{000, 011, 101, 110\}.$$

*El dígito que se agrega recibe el nombre de **dígito de control de paridad**. De igual manera si recibimos 111, como no está en C_3, entonces ha ocurrido un error en la transmisión, verificando cada una de las palabras de código, podemos concluir que las palabras de código 011, 101 y 110, solo necesitan un cambio en sus dígitos para que formen la palabra recibida. En este caso como hay más de una opción no podemos corregir, pero a medida avancemos en la teoría describiremos un mecanismo que nos permitan decidir cuál es la palabra más probable de haber sido enviada.*

Observación: En este último código, solo tiene eficiencia para cuando ocurre un error; en otras palabras los códigos con estructura parecida a la de C_3, solo tienen capacidad para detectar y corregir un error, y solo es capaz de detectar un número impar de errores, pero no de corregir. Esto se debe a que si ocurre un número par de cambios, esto no los podrá detectar el dígito de paridad, pues éste controla la cantidad de unos, si hay un número impar o par de dicho dígito.

Ejemplo 1.2.4. *En la palabra* 111*, hay un error porque el último dígito dice que hay un número impar de unos en los primeros dos espacios, pero no es cierto, hay dos, por eso podemos detectar un error, pero si ahora volvemos a cambiar otro dígito y se genera la siguiente palabra* 101*, en esta palabra han ocurrido dos errores y como vemos el dígito de paridad, en este caso si está acorde porque solo hay un uno, en los dos primeros espacios.*

Pero si agregamos dígitos para que podamos saber cuándo ocurre un error o no, y de hecho si eso nos permite hacer correcciones, ¿Afectará que este incremento de dígitos afecta en algo nuestra comunicación?, para saber esto veremos como medir que tanta información podemos agregar a una palabra de código y que tanto lleva como información extra para que pueda saber que se ha producido un error.

1.2.1. Tasa de información

Anteriormente verificamos que al agregar dígitos a cada palabra de código, el código adquiere la propiedad de poder detectar errores, así también agregando la cantidad indicada, logra también adquirir la facultad de corregir los errores ocurridos en la transmisión de los mensajes.

En este punto, si logramos dotar a nuestro código de la propiedad de detectar errores tendremos que cada una de las palabras tendrán dígitos que no llevan información, que se conoce como **redundancia**, la cual es necesaria para que el código pueda verificar, si la palabra sufrió un cambio en la transmisión del mensaje; para medir esta proporción de información y redundancia, veamos la siguiente definición:

Definición 1.2.5. *Llamaremos tasa de información (o simplemente **razón**) de un código, a un número que mide la proporción de cada palabra de código que es información llevada por el mensaje.*

La tasa de información de cada código $\mathbf{C}$ de longitud n es definido por (para el código binario):

$$\frac{1}{n}\log_2|\mathbf{C}|$$

Ya que podemos suponer que $1 \leq |\mathbf{C}| \leq 2^n$, es claro que el rango de la tasa de información esta entre 0 y 1; es así pues si aplicamos logaritmos a la desigualdad tendremos que $\log_2(1) \leq \log_2|\mathbf{C}| \leq \log_2 2^n \Rightarrow 0 \leq \frac{1}{n}\log_2|\mathbf{C}| \leq 1$, como el código mínimo tiene dos elementos, por tanto $n \geq 1$ por eso podemos dividir por n.

Por ejemplo, la tasa de información de $\mathbf{C_1}, \mathbf{C_2}$ y $\mathbf{C_3}$, de los ejemplos 1.2.1, 1.2.2 y 1.2.3, son $1, 1/3$ y $2/3$ respectivamente. Cada una de estas tasas de información parecen relacionarse con su respectivo código, ya que en el primero dice que toda la palabra es información, cosa que cambia en el segundo que dice que un tercio de la palabra es información, o sea 2 de cada seis dígitos se puede considerar información y el resto (4 dígitos) son redundancia (ya que podemos ver $1/3$ como $2/6$) y de igual forma para el último código donde

dos dígitos son información y uno de redundancia.

En resumen la tasa de información nos dice qué porcentaje de nuestro mensaje lleva información y cuánto lleva de redundancia.

Lo anterior nos permite saber qué tanto puede soportar nuestro código información o qué tanto podemos considerar como información valiosa, de hecho es claro que entre más dígitos tenga es peor pues si consideramos el siguiente caso:

Supongamos que todas las 2^{11} palabras de longitud 11 y el canal de confiabilidad $p = 1 - 10^{-8}$ y suponga que los dígitos transmitidos son de una tasa de 10^7 por segundo. Entonces la probabilidad de que una palabra sea transmitida incorrectamente es aproximadamente $11p^{10}(1-p)$, el cual es $11/10^8$. Así

$$\frac{11}{10^8}.\frac{10^7}{11} = 0{,}1$$

Lo que significa que 0,1 palabras por segundo son transmitidas incorrectamente sin ser detectadas. Esto es una palabra incorrecta cada 10 segundos, esto no es nada bueno ya que al pasar un minuto habrán ocurrido 6 palabras incorrectas, eso sucede ya que la palabra es un tanto larga, si las palabras fueran más cortas se reduciría este fenómeno. Este hecho nos dice que no debemos agregar dígitos al azar, esperando que esto arregle nuestro código, sino más bien poder encontrar un punto medio donde agregar dígitos no sea tan dañino como para que muchas palabras incorrectas pasen por minuto, con esto en mente veamos, la siguiente sección.

1.3. ¿Cómo encontrar la palabra de código más probable que fue envíada?

Es claro que cuando recibimos una palabra y dicha palabra no pertenezca a nuestro código, la palabra original en el proceso de su transmisión sufrió cambios. En ese sentido es necesario tener una herramienta o forma que nos permita de alguna manera poder predecir la palabra del código más probable que fue enviada.

Para poder deducir nuestra primera herramienta, consideremos primero que el ruido afecta a cada dígito de manera independiente y aleatoriamente, además consideremos palabras de longitud n. Por último, recordemos que nuestro canal **BSC** tiene la probabilidad p de confianza.

Así sea w la palabra recibida y sea v una palabra de nuestro código, ambas palabras de longitud n, y que estos difieren en d posiciones en sus dígitos, así como cada dígito puede

cambiar o mantenerse, en ese sentido podemos aplicar la distribución de Bernulli, a cada dígito y así tendremos que **la probabilidad de que** v **fue envíado, cuando** w **se recibe, es**:

$$\phi_p(w,v) = p^{n-d}(1-p)^d.$$

Para entender, la fórmula anterior veamos algunos ejemplos:

Ejemplo 1.3.1. *Calcular* $\phi_p(v,w)$ *con* $p = 0{,}97$*, para cada una de los siguientes pares de* v *y* w*.*

a) $v = 01101101$, $w = 10001110$.

Solución

De los códigos anteriores podemos concluir que $n = 8$ y $d = 5$, así

$$\phi_{0,97}(v,w) = (0{,}97)^{8-5}(1-0{,}97)^5 = 2{,}2178 \times 10^{-8}.$$

b) $v = 10110$, $w = 01001$.

Solución

Dados v y w, podemos concluir, $n = 5$ y $d = 5$, entonces

$$\phi_{0,97}(v,w) = (0{,}97)^{5-5}(1-0{,}97)^5 = 2{,}43 \times 10^{-8}.$$

Así es claro que si queremos saber cual palabra v del código es la más probable es necesario aplicar la formula a todas las palabras de código y seleccionar la palabra que dé la probabilidad máxima.

Observación: Por lo dicho anteriormente podemos decir que v **fue enviada cuando se recibe** w **si**:

$$\phi_p(w,v) = \text{máx}\{\phi_p(w,u) : u \in C\}.$$

EL proceso se vuelve muy engorroso y poco práctico, pues hay que saber en cuántos dígitos las palabras no concuerdan y además de calcular mediante la fórmula las probabilidades de cada uno de las palabras de código con la palabra recibida. El siguiente teorema nos permite tomar dicha decisión de una manera más práctica:

Teorema 1.3.2. *Suponga que tenemos* ***BSC*** *con* $1/2 < p < 1$*. Sean* v_1 *y* v_2 *códigos palabras y* w *una palabra, cada uno de longitud* n*. Suponga que* v_1 *y* w *no coinciden en* d_1 *posiciones y* v_2 *y* w *en* d_2 *posiciones. Entonces*

$$\phi_p(v_1,w) \leq \phi_p(v_2,w) \text{ si y sólo si } d_1 \geq d_2.$$

Demostración.

$$\begin{aligned}\phi_p(v_1, w) \leq \phi_p(v_2, w) &\Leftrightarrow p^{n-d_1}(1-p)^{d_1} \leq p^{n-d_2}(1-p)^{d_2} \\ &\Leftrightarrow \left(\frac{p}{1-p}\right)^{d_2-d_1} \leq 1 \\ &\Leftrightarrow d_2 - d_1 \leq 0, \text{ ya que } \frac{p}{1-p} > 1.\end{aligned}$$

□

El resultado anterior nos hace más fácil la selección de la palabra v del código, pues basta con revisar en cuántos dígitos no concuerdan con la palabra w recibida con las palabras v, del código, y luego de saber este dato, la palabra más probable será aquella que tenga menos discrepancia con respecto a la palabra recibida.

Para entender mejor lo anterior, veamos algunos ejemplos:

Ejemplo 1.3.3. *Suponga que* $w = 0010110$ *es recibido en un BSC con confiabilidad* $p = 0{,}90$*, ¿cuál de las siguientes palabras de códigos es la más probable de haber sido enviada?*
1001011, 1111100, 0001110, 0011001, 1101001.

Solución
Construimos una tabla donde se muestre que para cada palabra de código cuántos dígitos no concuerdan con w.

v	d (número de discordancia con w)
1001011	5
1111100	4
0001110	2
0011001	4
1101001	7

Por el teorema 1.3.2 la palabra más probable es la que tiene menos cambios y esta es 0001110, solo dos cambios.

Ejemplo 1.3.4. *Si* $C = \{01000, 01001, 00011, 11001\}$ *y la palabra* $w = 10110$ *es recibida, ¿cuál es la palabra de código más probable de haber sido enviada?.*

Solución

v	d (número de discordancia con w)
01000	4
01001	5
00011	3
11001	4

Por el teorema 1.3.2, la palabra más probable de haber sido envíada es 00011, al solo tener 3 cambios.

Nota: En el teorema 1.3.2 se asume que $1/2 < p < 1$. ¿Qué cambiaría en el enunciado del teorema 1.3.2, si reemplazamos esta suposición con:

a) $0 < p < 1/2$?

Solución

$$\begin{aligned}
\phi_p(v_1, w) \leqslant \phi_p(v_2, w) &\Leftrightarrow p^{n-d_1}(1-p)^{d_1} \leqslant p^{n-d_2}(1-p)^{d_2} \\
&\Leftrightarrow p^n p^{-d_1}(1-p)^{d_1} \leqslant p^n p^{-d_2}(1-p)^{d_2} \\
&\Leftrightarrow \cancel{p^n} p^{-d_1}(1-p)^{d_1} \leqslant \cancel{p^n} p^{-d_2}(1-p)^{d_2} \\
&\Leftrightarrow p^{-d_1}(1-p)^{d_1} \leqslant p^{-d_2}(1-p)^{d_2} \\
&\Leftrightarrow p^{-d_1+d_2}(1-p)^{d_1-d_2} \leqslant 1 \\
&\Leftrightarrow \frac{p^{d_2-d_1}}{(1-p)^{d_2-d_1}} \leqslant 1 \\
&\Leftrightarrow \left(\frac{p}{1-p}\right)^{d_2-d_1} \leqslant 1 (*)
\end{aligned}$$

Además por hipótesis general tenemos que $0 < p < 1/2 \Rightarrow p < 1 - p$, de esta última desigualdad solo pasamos a dividir por $1 - p$, entonces tendremos $\frac{p}{1-p} < 1$. Teniendo en cuenta esta última desigualdad es claro que la desigualdad $(*)$ se garantiza si y solo si el exponente de $\frac{p}{1-p}$ cumpla ser mayor o igual que cero, por lo tanto:

$$d_2 - d_1 \geq 0 \Leftrightarrow d_2 \geq d_1$$

Así el resultado del teorema 1.3.2 queda de la siguiente manera:

$$\phi_p(v_1, w) \leqslant \phi_p(v_2, w) \Leftrightarrow d_1 \leqslant d_2$$

Lo que quiere decir que la palabra más probable es la que más discrepancia tenga con respecto a la palabra recibida.

b) $p = 1/2$

Solución

En este caso observemos que para cualesquiera v y w palabras de códigos, donde n es el tamaño de v y w, y d, la cantidad de dígitos donde no concuerdan las dos palabras, así:

$$\phi_{1/2}(v, w) = (1/2)^{n-d}(1 - 1/2)^d = (1/2)^{n-d}(1/2)^d = (1/2)^{n-d+d} = (1/2)^n$$

Lo que quiere decir que no importa que palabras comparemos siempre dará $(1/2)^n$, lo que significa que cualquier palabra de código tiene la misma probabilidad de ser la palabra enviada.

Es claro que el método funciona siempre que no existan dos palabras que tengan el mismo valor d (recuerden que el número d, representa la cantidad de dígitos en que la palabra recibida concuerda con una palabra del código), en este caso, no podemos concluir de manera efectiva, ya que existen dos o más candidatos a ser la palabra enviada. Será necesario desarrollar elementos más potentes en nuestros código, que permita que casos como estos sean superados, esto lo veremos más adelante a medida avancemos en nuestro estudio.

1.4. El espacio vectorial de todas las palabras de longitud n

Por la forma que hemos venido trabajando las palabras, esto se ha hecho como si tuviéramos vectores, con n coordenadas y coordenada a coordenada sumamos. Esto quiere decir que si definimos adecuadamente nuestra operación suma y producto por escalar en el conjunto de todas las palabras de longitud n, tendremos un espacio vectorial.

Primero estudiemos la estructura de $K = \{0, 1\}$, definimos la suma dentro de K y producto de K (pensando en una suma y producto módulo 2), de esta manera tenemos:

1. $0 + 0 = 0$
2. $0 + 1 = 1 + 0 = 1$
3. $1 + 1 = 0$
4. $0 * 0 = 0$

5. $0 * 1 = 1 * 0 = 0$
6. $1 * 1 = 1$

Ahora tomemos el producto K, n veces, así:

$$\underbrace{K \times K \times ... \times K}_{n-\text{veces}} = K^n$$

Esto hace que K^n los elementos son n-adas de ceros y unos, por ejemplo:

- $(1, 1, 0, ..., 1)$ y $(0, 1, 0, ..., 0)$.

Pero para efectos de no cambiar la forma que hemos venido tomando las palabras, tomaremos cada n-ada, como la secuencia de dígitos que las componentes forman, así para los ejemplos anteriores tenemos:

- $(1, 1, 0, ..., 1) = 110...1$ y $(0, 1, 0, ..., 0) = 010...0$

Como cada n-ada, tiene n componentes, la secuencia de dígitos que estos forman tienen también n elementos, esto hace que no genere ningún problema con las definiciones que al inicio se habían dado.

Además al tener K^n, definimos de manera natural la suma y producto por escalar sobre K.

1. La suma en K^n, se definirá componente a componente y haciendo uso de las operaciones en listadas para K (las primeras tres para ser precisos) por cada componente.

   Ejemplo 1.4.1. *Para el caso de* $n = 3$, $101 + 111 = (1 + 1, 0 + 1, 1 + 1) = 010 \Rightarrow 101 + 111 = 010$.

2. El producto por escalar de K^n sobre K, de igual manera se define componente a componente usando lo definido para K.

   Ejemplo 1.4.2. *Consideremos de nuevo* $n = 3$, $1 * (101) = (1 * 1, 1 * 0, 1 * 1) = 101 \Rightarrow 1 * (101) = 101$.

Usando las definiciones anteriores, podemos demostrar que K^n es un espacio vectorial sobre K.

Sean $u, v, w \in K^n$ y $a, b \in K$, con $u = u_1u_2...u_n$, $v = v_1v_2...v_n$ y $w = w_1w_2...w_n$, donde $\forall i, u_i, v_i, w_i \in K$, así:

1. Es trivial que $u + v \in K^n$; ya que se definió de manera natural a partir de las operaciones de K.

2. Verifiquemos: $(u + v) + w = u + (v + w)$.

$$\begin{aligned}(u_1u_2\dots u_n + v_1v_2\dots v_n) + w_1w_2\dots w_n &= (u_1 + v_1, u_2 + v_2, \dots, u_n + v_n) + w_1w_2\dots w_n \\ &= (u_1 + v_1 + w_1, u_2 + v_2 + w_2, ..., u_n + v_n + w_n) \\ &= (u_1 + (v_1 + w_1), u_2 + (v_2 + w_2), ..., u_n + (v_n + w_n)) \\ &= u_1u_2...u_n + (v_1 + w_1, v_2 + w_2, ..., v_n + w_n) \\ &= u + (v + w).\end{aligned}$$

3. Existencia de elemento identidad en la suma, es decir $u + \mathbf{0} = \mathbf{0} + u = u$, donde $\mathbf{0}$ es la palabra cero.

$$\begin{aligned}u + \mathbf{0} = u &\Leftrightarrow u_1u_2\dots u_n + 0_10_2\dots 0_n = u_1u_2\dots u_n \\ &\Leftrightarrow (u_1 + 0_1, u_2 + 0_2, ..., u_n + 0_n) = u_1u_2\dots u_n \\ &\Leftrightarrow u_i + 0_i = u_i, \forall i \\ &\Leftrightarrow 0_i = 0, \forall i, \text{ ya que } K \text{ solo tiene como elemento neutro de la suma al cero} \\ &\Leftrightarrow \mathbf{0} \text{ es la palabra cero.}\end{aligned}$$

   Análogo para el otro caso.

4. Para algún $u' \in K^n$, tenemos $u + u' = u' + u = \mathbf{0}$.

$$\begin{aligned}u + u' = \mathbf{0} &\Leftrightarrow u_1u_2\dots u_n + u'_1u'_2\dots u'_n = 00\dots 0 \\ &\Leftrightarrow (u_1 + u'_1, u_2 + u'_2, \dots, u_n + u'_n) = 00\dots 0 \\ &\Leftrightarrow u_i + u'_i = 0, \forall i \\ &\Leftrightarrow u'_i = u_i, \forall i, \text{ ya que en } K \text{ cada elemento es su mismo inverso aditivo} \\ &\Rightarrow u' = u \text{ el inverso aditivo de una palabra es ella misma.}\end{aligned}$$

   Análogo para el otro caso.

5. Verifiquemos $v + w = w + v$, para $w, v \in K^n$.

$$\begin{aligned}v_1v_2...v_n + w_1w_2...w_n &= (v_1 + w_1, v_2 + w_2, ..., v_n + w_n) \\ &= (w_1 + v_1, w_2 + v_2, ..., w_n + v_n) \\ &= w_1w_2...w_n + v_1v_2...v_n \\ &\Rightarrow v + w = w + v.\end{aligned}$$

6. Es evidente que $av \in K^n$, para $a \in K$ y $v \in K^n$, ya que se definió de manera natural usando la operación de K.

7. Sean $a \in K$, $v, w \in K^n$, entonces se cumple que: $a(v+w) = av + aw$.

$$\begin{aligned} a(v+w) &= a(v_1v_2...v_n + w_1w_2...w_n) \\ &= a((v_1+w_1, v_2+w_2, ..., v_n+w_n)) \\ &= (a(v_1+w_1), a(v_2+w_2), ..., a(v_n+w_n)) \\ &= (av_1+aw_1, av_2+aw_2, ..., av_n+aw_n) \\ &= av_1av_2...av_n + aw_1aw_2...aw_n \\ &= a(v_1v_2...v_n) + a(w_1w_2...w_n) \\ &= av + aw. \end{aligned}$$

8. Demostremos que se cumple $(a+b)v = av + bv$, para $a, b \in K$ y $v \in K^n$.

$$\begin{aligned} (a+b)v &= (a+b)(v_1v_2...v_n) \\ &= ((a+b)v_1, (a+b)v_2, ..., (a+b)v_n) \\ &= (av_1+bv_1, av_2+bv_2, ..., av_n+bv_n) \\ &= av_1av_2...av_n + bv_1bv_2...bv_n \\ &= a(v_1v_2...v_n) + b(v_1v_2...v_n) \\ &= av + bv. \end{aligned}$$

9. Veamos $(ab)v = a(bv)$, para $a, b \in K$ y $v \in K^n$.

$$\begin{aligned} (ab)v &= (ab)v_1v_2...v_n \\ &= ((ab)v_1, (ab)v_2, ..., (ab)v_n) \\ &= (a(bv_1), a(bv_2), ..., a(bv_n)) \\ &= a((bv_1, bv_2, ..., bv_n)) \\ &= a(b(v_1v_2...v_n)) \\ &= a(bv). \end{aligned}$$

10. Verifiquemos $1v = v$, $\forall v \in K^n$ y $1 \in K$.
    $1v = 1(v_1v_2...v_n) = (1v_1, 1v_2, ..., 1v_n) = v_1v_2...v_n \Rightarrow 1v = v$.

Por lo tanto podemos concluir que K^n es un espacio vectorial sobre K y a dicho espacio vectorial lo denotaremos como: K^n o $V(n,2)$, donde n representa la longitud de las palabras además de ser la dimensión del espacio vectorial, y el 2, solo hace referencia a que el campo K solo está formado por dos elementos, que en nuestro caso son cero y uno (en otras palabras $K = \mathbb{Z}/2\mathbb{Z}$).

Observemos unas propiedades importantes que cumple $V(n,2)$.

Proposición 1.4.3. *Toda palabra de longitud n, multiplicada por el escalar cero de K, es la palabra cero de K^n.*

Demostración. Sea $0 \in K$, y $v \in K^n$, tenemos:

$$0v = 0(v_1v_2...v_n) = (0v_1, 0v_2, ..., 0v_n) = 00, ..., 0 \Rightarrow 0v = \mathbf{0}$$

□

Proposición 1.4.4. *Si v y w son palabras en K^n y $v + w = \mathbf{0}$, entonces $v = w$.*

Demostración. Sea $w, v \in K^n$, tal que $v + w = \mathbf{0}$, ahora sumamos en ambos lados de la igualdad w y usando el ejercicio anterior junto con las propiedades definidas en la sección 1.7, tendremos

$$v + w = \mathbf{0} \Rightarrow v + w + w = \mathbf{0} + w \qquad \Rightarrow v + \mathbf{0} = w \Rightarrow \therefore v = w$$

□

Proposición 1.4.5. *Si u, v y w son palabras en $V(n, 2)$ y $u + v = w$, entonces $u + w = v$.*

Demostración. Sean $u, v, w \in K^n$, tal que $u + v = w$, basta con sumar u a ambos lados de la igualdad y tendremos el resultado deseado.

$\Rightarrow u + v + u = w + u$
$\Rightarrow (u + u) + v = w + u$, por la propiedad 5 y 2
$\Rightarrow \mathbf{0} + v = w + u$, por la propiedad 4
$\Rightarrow v = w + u$ por la propiedad 3

Por lo tanto $u + w = v$. □

De estas propiedades podemos obtener algo nuevo para nuestro estudio. Por ejemplo, si mandamos una palabra v en un canal **BSC**, y recibimos w, es claro que cuando tomemos la suma de ambas palabras $v + w$, tendremos que los ceros que éste tenga en sus componentes, representarán los dígitos donde estos concuerdan, y por ende los unos presentarán las posiciones donde los dígitos no concuerdan; ya que por la propiedad 4 del espacio vectorial $V(n, 2)$, notamos que cada palabra en el espacio vectorial es su propio inverso, por esta razón cuando sumamos dos palabras, los ceros representan donde los dígitos son iguales y los unos donde no se parecen. Esto quiere decir que si tomamos $u = v + w$, podemos recuperar la palabra v, a partir de u y de w, ya que si $u = v + w$, usando la proposición 1.4.5, podemos concluir $v = u + w$; a esta nueva palabra la conoceremos como **patrón de error**, así tenemos la siguiente definición:

Definición 1.4.6. *Sea w una palabra recibida y sea v una palabra de nuestro código, llamaremos **patrón de error** u, a la palabra formada por la suma de w y v ($u = v + w$).*

1.5. El espacio métrico de los códigos

En este apartado, veremos que nuestro conjunto K^n, lo podemos dotar de una métrica y que esto hará que el par (K^n, d) sea un espacio métrico, donde d será la métrica. Nuestro objetivo en esta sección es buscar la métrica adecuada para el conjunto K^n.

Primero comencemos por definir *el peso de una palabra*, este concepto recibe el nombre de **peso de Hamming**, en honor al matemático **Richard Hamming**.

Definición 1.5.1. *El **peso de Hamming**, o simplemente **peso**, de una palabra v de longitud n, es la cantidad de unos que esté posee y se denotará por $wt(v)$.*

Observación: En la definición 1.5.1, el peso de la palabra es la cantidad de unos que ésta tenga, una forma alternativa de definir el peso de una palabra es como la suma de sus dígitos; dicha suma se hace sobre los reales, así:

$$\forall v \in K^n : wt(v) = wt(a_1, a_2, ..., a_n) = \sum_{i=1}^{n} a_i$$

Ejemplo 1.5.2. *Así por ejemplo para la palabra* 110101*, su peso es* $wt(110101) = 4$ *y para* 00000 *es* $wt(00000) = 0$.

El peso de Hamming cumple una lista de propiedades que serán útiles y necesarias para nuestros fines.

Proposición 1.5.3. *Para $a \in K$ y $w, v \in K^n$, el peso de Hamming, cumple las siguientes propiedades:*

1. $0 \leq wt(v) \leq n$
2. $wt(v) = 0$ *si y solo si* $v = \boldsymbol{0}$
3. $wt(v + w) \leq wt(v) + wt(w)$
4. $wt(a.v) = a.wt(v)$

Nota: A lo largo de la demostración usaremos la definición alterna que se dió del peso de una palabra, realizando la suma de sus dígitos sobre los números reales. En el caso que sea necesario hacer la suma en K de los dígitos de la palabra, lo haremos explícito colocando mód(2).

Demostración.

1. $0 \leq wt(v) \leq n$

   Sea $v \in K^n$, entonces $v = (a_1, a_2, ..., a_n)$, donde $a_i \in K, \forall i$, como cada $a_i \in K$ y $K = \{0, 1\}$, entonces $a_i \geq 0, \forall i$, esto implica que:

$$\sum_{i=1}^{n} a_i \geq 0, \text{ ya que } a_i \geq 0, \forall i$$

Y por definición tendremos

$$wt(v) = \sum_{i=1}^{n} a_i \geq 0 \tag{1.1}$$

Por otro lado como $a_i \in K$, eso quiere decir que cada $a_i \leq 1$, entonces tendremos:

$$wt(v) = \sum_{i=1}^{n} a_i \leq \sum_{i=1}^{n} 1 = n \tag{1.2}$$

De (1.1) y (1.2), tenemos $0 \leq wt(v) \leq n$.

2. $wt(v) = 0$ si y solo si $v = \mathbf{0}$

   Observemos primero que si cada $a_i \geq 0 \Rightarrow \sum_{i=1}^{n} a_i \geq 0$. Con esto pasemos a la demostración.

   Como es una doble implicación trabajemos por partes.

   ($\Rightarrow$) Sea $v \in K^n$, tal que $wt(v) = 0$, donde $v = (a_1, a_2, ..., a_n)$, entonces por definición tendremos:

   $$wt(v) = \sum_{i=0}^{n} a_i = 0$$

   Ahora tomando en cuenta la observación del inicio tendremos

   $$0 \leq \sum_{i=1}^{n} a_i = 0 \Rightarrow a_i = 0, \forall i$$

   Esto quiere decir que v es la palabra de código cero (ya que todos sus dígitos son cero), por lo tanto $v = \mathbf{0}$.

   ($\Leftarrow$) Sea $v \in K^n$, tal que $v = \mathbf{0}$, donde $v = (0, 0, ..., 0)$, ahora usando la definición de peso, tendremos:

   $$wt(v) = \sum_{i=1}^{n} a_i = \sum_{i=1}^{n} 0 = 0$$

   Por lo tanto $wt(v) = 0$, y con esto se completa la prueba.

3. $wt(v+w) \leq wt(v) + wt(w)$

   Sabemos que no existen dos dígitos $a, b \in K$, tal que $a + b = 0$ y $a + b = 1$.

   Teniendo en cuenta lo anterior, también observemos que el peso para cualesquiera $v, w \in K^n$, donde $v = (a_1, a_2, ..., a_n)$ y $w = (b_1, b_2, ..., b_n)$, podemos expresarlo como la suma de dos sumatorios;

$$wt(v+w) = \sum_{a_i+b_i=0} (a_i + b_i \text{ mód}(2)) + \sum_{a_i+b_i=1} (a_i + b_i \text{ mód}(2))$$

   Donde el primer sumatorio toma los ceros de la palabra resultante de la suma de v y w, y el segundo cuenta los unos que tiene dicha palabra.

   Como última observación veamos las siguientes relaciones:

$$\sum_{a_i+b_i=0} (a_i + b_i \text{ mód}(2)) \leq \sum_{a_i+b_i=0} (a_i) + \sum_{a_i+b_i=0} (b_i) \tag{1.3}$$

$$\sum_{a_i+b_i=1} (a_i + b_i \text{ mód}(2)) = \sum_{a_i+b_i=1} (a_i) + \sum_{a_i+b_i=1} (b_i) \tag{1.4}$$

   Para la primera desigualdad basta ver que si $a_i + b_i = 0$ mód (2), entonces $a_i = b_i = 0$ mód (2) 0 $a_i = b_i = 1$ mód (2), Si todos cumplen $a_i = b_i = 0$ mód (2), entonces tendremos la igualdad porque estaríamos sumando ceros y si existe al menos un i tal que $a_i = b_i = 1$ mód (2), tendremos que la suma del lado derecho tiene al menos un uno por suma y esto ya vuelve mayor que cero; ya que del lado izquierdo la suma siempre da cero.

   Por otro lado en el caso que $a_i + b_i = 1$ mód (2), esto se da ya sea porque $a_i = 0$ mód (2) y $b_i = 1$ mód (2) o $a_i = 1$ mód (2) y $b_i = 0$ mód (2), en cualquiera de los dos casos no cambiaría nada porque en uno de los sumatorios, la suma da cero y en el otro sumatorio uno, entonces por cada uno que se sume en la izquierda también se suma en la derecha (solamente una vez), por esta razón la segunda desigualdad también es cierta.

   Así finalmente usando todo lo anterior, pasemos a demostrar:

$$wt(v+w) \leq wt(v) + wt(w).$$

   Sean $v, w \in K^n$, donde $v = (a_1, a_2, ..., a_n)$ y $w = (b_1, b_2, ..., b_n)$

$$\begin{aligned}
wt(v+w) &= \sum_{a_i+b_i=0} (a_i + b_i \text{ mód}(2)) + \sum_{a_i+b_i=1} (a_i + b_i \text{ mód}(2)) \\
&= \sum_{a_i+b_i=0} (a_i + b_i \text{ mód}(2)) + \sum_{a_i+b_i=1} (a_i) + \sum_{a_i+b_i=1} (b_i); \text{ por (1.3)} \\
&\leq \sum_{a_i+b_i=0} (a_i) + \sum_{a_i+b_i=0} (b_i) + \sum_{a_i+b_i=1} (a_i) + \sum_{a_i+b_i=1} (b_i); \text{ por (1.4)} \\
&= \sum_{a_i+b_i=0} (a_i) + \sum_{a_i+b_i=1} (a_i) + \sum_{a_i+b_i=0} (b_i) + \sum_{a_i+b_i=1} (b_i) \\
&= \sum_{i=1}^{n} a_i + \sum_{i=1}^{n} b_i \\
&= wt(v) + wt(w)
\end{aligned}$$

Por lo tanto se cumple: $wt(v+w) \leq wt(v) + wt(w)$.

4. $wt(a.v) = a.wt(v)$

   Sean $a \in K$ y $v \in K^n$, utilicemos las operaciones definidas en K y K^n, así analicemos por casos, en función de los valores que puede tomar a:

   (1) Si $a = 0$, entonces tendremos

   $$wt(a.v) = wt(0.v) = wt(\mathbf{0}) = 0 = 0.wt(v) = a.wt(v).$$

   (2) Si $a = 1$, entonces

   $$wt(a.v) = wt(1.v) = wt(v) = 1.wt(v) = a.wt(v).$$

   Y en todos los casos se cumple, por lo tanto hemos demostrado $\forall a \in K, \forall v \in K^n$, se cumple $wt(a.v) = a.wt(v)$.

□

Observación: Como podemos ver el peso de Hamming tiene propiedades muy parecidas a las de una métrica, de hecho lo que hemos demostrado en la proposición anterior, es que $wt(v)$ es una norma y por tanto nuestro espacio vectorial K^n no solo es un espacio vectorial, sino también un **espacio vectorial normado**, con norma $wt(\cdot)$ que cumple:

1. **No negatividad**, esto nos dice que $wt(\cdot)$ es mayor o igual que cero es siempre positivo. Además solo da cero cuando tenemos la palabra cero (”vector cero”).

2. **Desigualdad triangular**; esto es sumamente importante.

3. **Homogeneidad**, nos dice el peso de una palabra múltiplo de otra, su peso será el peso de la palabra multiplicada por el escalar.

Definición 1.5.4. *El espacio vectorial K^n, con el peso de Hamming $wt(\cdot)$, es un **espacio vectorial normado**. En otras palabras el par $(K^n, wt(\cdot))$, es un espacio vectorial normado.*

Como sabemos de nuestros cursos de análisis, todo espacio normado puede ser convertido a un espacio métrico, con la métrica inducida por la norma. a esta métrica que induce el peso de Hamming, recibe el nombre de **distancia de Hamming** o simplemente distancia; con esto tenemos la siguiente definición:

Definición 1.5.5. *La **distancia de Hamming** o **distancia** entre dos palabras v y w, se define como $d(v,w) = wt(v+w)$; lo que significa que es el número de dígitos que no comparten en la misma posición.*

Ejemplo 1.5.6. *Encontremos la distancia que hay entre las palabras* 01011 *y* 00111*, aplicando la definición tendremos $d(01011, 00111) = wt(01011 + 00111) = wt(01100) = 2$ y en el caso que se desee saber la distancia entre la misma palabra tendremos $d(01011, 01011) = wt(01011 + 01011) = wt(00000) = 0$.*

Al igual que el peso, éste cumple ciertas propiedades (que es más que verificar, que en efecto es una métrica $d(\cdot,\cdot)$).

Proposición 1.5.7. *Sea w,v y u, palabras de longitud n, entonces la distancia de Hamming, cumple las siguientes propiedades:*

1. $0 \leq d(v,w) \leq n$
2. $d(v,w) = 0$ *si y solo si* $v = w$
3. $d(v,w) = d(w,v)$
4. $d(v,w) \leq d(v,u) + d(u,w)$
5. $d(av,aw) = a.d(v,w)$

Demostración. Analicemos cada caso, para demostrar las propiedades.

- $0 \leq d(v,w) \leq n$

  Sean $v, w \in K^n$, entonces tendremos por definición de distancia,

  $$d(v,w) = wt(v+w)$$

  y aplicando la propiedad (1) del peso, tendremos:

  $$0 \leq wt(v+w) \leq n \Rightarrow 0 \leq d(v,w) \leq n$$

  Lo cual completa la demostración.

- $d(v,w) = 0$ si y solo si $v = w$

  Sean $v, w \in K^n$, entonces tendremos, que la distancia entre estas dos palabras de código es:

  $$d(v,w) = wt(v+w).$$

  Como la demostración es una doble implicación, entonces analizamos en dos sentidos:

  ($\Rightarrow$) Ya que $d(v,w) = wt(v+w) = 0$, entonces por la propiedad (2) del peso, tendremos;

  $$wt(v+w) = 0 \Rightarrow v + w = \mathbf{0},$$

  Ahora usando la proposición 1.4.4 se concluye $v = w$.

  ($\Leftarrow$) Ahora como tenemos que $v + w = \mathbf{0}$, entonces hacemos los pasos anteriores pero en sentido contrario (ya que se tiene: $\forall v \in K^n : v + v = \mathbf{0}$, y la propiedad (2) del peso, es un si y solo si),

  $$\begin{aligned} v = w &\Rightarrow v + w = w + w \\ &\Rightarrow v + w = \mathbf{0} \\ &\Rightarrow wt(v+w) = 0 \\ &\Rightarrow d(v,w) = 0. \end{aligned}$$

  Lo que completa la prueba.

- $d(v,w) = d(w,v)$

  Primero recordemos que la operación definida sobre K^n es simétrica, lo que significa que $v + w = w + v$.

  Teniendo en cuenta lo anterior, comencemos la demostración.

  $$d(v,w) = wt(v+w) = wt(w+v) = d(w,v).$$

  Por lo tanto $d(v,w) = d(w,v)$.

- $d(v,w) \leq d(v,u) + d(u,w)$
  Sean $v, u, w \in K^n$,

$$\begin{aligned} d(v,w) &= wt(v+w) \\ &= wt(v+w+\mathbf{0}) \quad \text{Sumamos la palabra cero convenientemente,} \\ &= wt(v+w+u+u) \quad \text{ya que } u+u=\mathbf{0}, \\ &= wt(v+u+u+w) \quad \text{Porque la suma en } K^n \text{ es conmutativa,} \\ &= wt((v+u)+(u+w)) \quad \text{Propiedad asociativa,} \\ &\leq wt(v+u)+wt(u+w) \quad \text{Usando la propiedad (3) del peso,} \\ &= d(v,u)+d(u,w). \end{aligned}$$

  Por lo tanto se ha demostrado $d(v,w) \leq d(v,u) + d(u,w)$.

- $d(av,aw) = ad(v,w)$
  Sean $a \in K$ y $v \in K^n$, aplicando el resultado (4) del peso y las propiedades de las operaciones en K^n, tendremos:

$$d(av,aw) = wt(av+aw) = wt(a(v+w)) = a.wt(v+w) = a.d(v,w).$$

  Por lo tanto se concluye $d(av,aw) = a.d(v,w)$.

□

Por lo tanto por la propiedad anterior tenemos que la distancia de Hamming es una métrica, y así el par (K^n, d) es un espacio métrico.

Lo grandioso que hemos llegado a demostrar es que nuestro espacio vectorial es más que un espacio normado sino también métrico, lo cual concuerda con lo estudiado en el curso de análisis matemático, donde nos enseñan que todo espacio normado, es un espacio métrico usando la métrica inducida por la norma.

1.6. Los dos problemas fundamentales de la teoría de códigos

A este punto hemos visto que si agregamos una cierta cantidad de dígitos (redundancia) a nuestras palabras de código, entonces podemos hacer que nuestro código adquiera la habilidad de detectar (hasta cierto punto) errores en la transmisión, pero también esto hace que nuestras palabras sean más grandes (en longitud), lo cual se traduce en mayor costo

de envío o transmisión, por ello es necesario solo agregar la cantidad adecuada de dígitos.

También debemos notar que no tenemos control sobre la probabilidad p, de nuestro canal BSC; lo que significa que no podemos controlar cuando un dígito es transmitido correctamente ni tampoco cuando no.

Por ello es necesario saber cuantas palabras necesitamos para comunicarnos, de hecho es más importante eso a saber cuantas palabras podemos enviar en el canal, ya que al no tener control sobre el ruido, tampoco tenemos certeza de cuantas palabras como máximo podemos mandar sin que estos presenten problemas. Por ello, saber cuantas palabras se necesitan para comunicarme es un paso muy importante, para diseñar un código eficiente y con la capacidad detectar y sobre todo corregir dichos errores.

Todo lo anterior nos permite concluir tres cosas:

1. Tenemos control sobre como enviar una palabra, así como elegir las palabras que se usarán para comunicarse.

2. Es imposible tener control de canal, en la forma en que este afecta a las palabras (el ruido es aleatorio en cada dígito).

3. Podemos detectar y corregir el error de transmisión, esto porque podemos elegir con anticipación un código con estas facultades (por ejemplo usando el bit de paridad).

Como vemos solo al inicio y al final del proceso de transmisión podemos de alguna manera controlar la comunicación, por ello los dos grandes problemas de los que hace referencia el título de esta sección son:

1. Codificación.

2. Descodificación.

Definición 1.6.1. ***Codificación*** *es el método que permite convertir un carácter de un lenguaje natural (como el de un alfabeto o silabario) en un símbolo de otro sistema de representación, como un número o una secuencia de pulsos eléctricos en un sistema electrónico.*

Lo principal en esto es saber *elegir* el código adecuado para mandar nuestros mensajes, esta selección debe hacerse de manera eficiente y en términos generales debe seguirse la siguiente lógica:

1. Primero debemos elegir la longitud necesaria para nuestras palabras (un entero k), y de todas estas palabras de longitud k, debemos elegir las palabras que usaremos para formar nuestro mensajes (esto será un conjunto M), así tenemos que el código deberá cumplir $|M| \leq |V(k,2)| = 2^k$.

2. Luego debemos elegir la cantidad necesaria de dígitos a agregar a cada palabra de nuestro conjunto M, de tal manera que M, tenga la habilidad de poder detectar o corregir errores, sea lo que nosotros necesitemos. Al hacer esta elección de agregar más dígitos a cada palabra de nuestro conjunto hace que nuestras palabras se conviertan en palabras de longitud n, y a este nuevo conjunto con palabras de longitud n (formadas por las palabras de nuestro conjunto M), conformará nuestro **código** C.

Definición 1.6.2. ***Descodificación*** *es el proceso en el cual el receptor transforma el código utilizado por el emisor al lenguaje natural. De esta forma los signos son asociados a las ideas que el emisor trató de comunicar.*

Normalmente recibimos una palabra $w \in K^n$, y la forma que hemos venido trabajando los códigos, es verificar cual palabra de nuestro código requiere los menores cambios para formar la palabra w, y esta idea de hecho tiene un nombre y se conoce como **descodificación de máxima probabilidad** o simplemente **MLD** (por sus siglas en inglés *maximum likelihood decoding*). Al aplicar este principio de la descodificación, tenemos dos formas de aplicarla, las cuales son las siguientes:

- **Descodificación completa de máxima probabilidad** o simplemente **CMLD** (por sus siglas en inglés *Complete Maximum Likelihood Decoding*), esta forma versa en la idea de que si solamente una palabra $v \in C$ está cerca de w, esto significa que

  $$\text{Si } d(v,w) < d(u,w), \forall u \in C : u \neq v, \text{ entonces } w \text{ se descodifica como } v.$$

  En el caso de que exista más de una palabra que está a una distancia mínima de w, entonces se elige arbitrariamente una de ellas y w se descodifica como esa palabra que elegimos.

- **Descodificación incompleta de máxima probabilidad** o simplemente **IMLD** (por sus siglas en inglés *Incomplete Maximum Likelihood Decoding*), la diferencia radical de esta forma a la CMLD es que cuando tengamos más de una palabra a una misma distancia mínima a w, entonces no elegimos arbitrariamente, sino que pedimos una retransmisión del mensaje, en caso contrario que solo haya una palabra de código más cercana a w, entonces IMLD actúa como CMLD.

Trabajaremos con IMLD, en el resto del trabajo. Es necesario advertir al lector que este enfoque no siempre funciona, ya que en caso de que en nuestro canal BSC, ocurran demasiados errores en la transmisión, entonces IMLD falla.

Es de notar que con MLD, al tener una única palabra v de nuestro código cercana a la palabra recibida w, podemos descodificar w como v, que matemáticamente es que la distancia $d(v,w)$ es la más pequeña cuando dejamos fijo w y variamos en los elementos de C. Además por definición tenemos que la distancia es el peso de la la suma de v con w, así $d(v,w) = wt(v+w)$, y de hecho en la sección donde estudiamos a K^n como espacio

vectorial, definimos que $v + w$, se llama patrón de error, esto significa que la palabra se puede descodificar, si el peso del patrón de error de w con v, es el más pequeño.

Por otro lado, demostramos que la palabra que tenga menos diferencias en sus dígitos con respecto la palabra recibida es la más probable de haber sido enviada, entonces lo curioso es que el patrón de error, su peso nos dice también eso, cuántas diferencias hay entre la palabra v y w, ya que cuando sumamos dichas palabras, en las posiciones donde tengan el mismo dígito dará cero y donde no, será uno, así al sacar el peso nos dice el número de unos que este posee y por ende cuántas diferencias hay entre dichas palabras.

Por lo tanto en el teorema 1.3.2 el número d es de hecho el peso de u patrón de error de w y v, así podemos reescribir nuestro teorema de la siguiente manera:

$$\phi_p(v_1, w) \leq \phi_p(v_2, w) \Leftrightarrow wt(v_1 + w) \geq wt(v_2 + w)$$

Lo que significa que *la palabra de código más probable es la que tenga un patrón de error de peso más pequeño*.

Como vemos IMLD concuerda con lo que ya habíamos demostrado, así la estrategia para aplicar IMLD, será obtener el patrón de error de todas las palabras de código con la palabra recibida y elegir la que tenga menor peso y la palabra de código que tenga el patrón de error de esta forma será usado para descodificar w. Con esto en mente veamos algunos ejemplos.

Ejemplo 1.6.3. *Sea* $|M| = 2, n = 3$, *y* $C = \{001, 101\}$, *Si enviamos la palabra* $v = 001$, *¿cuándo IMLD concluye correctamente que v fue enviada, y cuándo concluye incorrectamente que* 101 *fue enviado?*

Solución

Para aplicar IMLD vamos a construir una tabla, donde en la primera columna estarán todas las posibles palabras que podemos recibir. Luego agregamos tantas columnas como palabras tenga nuestro código (en nuestro caso es 2) y estas columnas tendrán los patrones de error de cada una de las palabras de código con la palabra recibida por fila. Finalmente agregamos una última columna donde pondremos la descodificación siguiendo el enfoque de selección dada por IMLD (seleccionando siempre el patrón de error de menor peso). En la tabla colocaremos con un asterisco al patrón de error que tenga esa característica y la palabra de código que generó este patrón es la que se seleccionará para descodificar w). Con ésto tendremos la siguiente tabla:

Recibido	Patrón de error		Descodificación
w	$001 + w$	$101 + w$	v
000	001*	101	001
001	000*	100	001
010	011*	111	001
011	010*	110	001
100	101	001*	101
101	100	000*	101
110	111	011*	101
111	110	010*	101

Por lo tanto IMLD concluye correctamente cuando recibe las palabras $000, 001, 010, 011$, pero en el caso que se reciba $100, 101, 110, 111$, IMLD concluirá incorrectamente que se envió 101.

Ejemplo 1.6.4. *Sea $|M| = 3, n = 3$. Para cada palabra $w \in K^3$ que se puede recibir, encontrar las palabras w para el cual IMLD concluye correctamente que solo una palabra del código $C = \{000, 001, 110\}$, fue enviada.*

Solución:

De igual manera que en el ejemplo anterior hacemos una tabla.

Recibido	Patrón de error			Descodificación
w	$000 + w$	$001 + w$	$110 + w$	v
000	000*	001	110	000
001	001	000*	111	001
010	010*	011	100*	-
011	011	010*	101	001
100	100*	101	010*	-
101	101	100*	011	001
110	110	111	000*	110
111	111	110	001*	110

Por lo tanto se puede concluir lo siguiente:

1. Si recibimos 000, este se puede descodificar como 000.
2. Si recibimos $001, 011$ y 101, estos son descodificados como 001.
3. Si recibimos 110 y 111, estos se descodifican como 110.
4. En los casos 010 y 100, se requiere una retransmisión ya que hay dos patrones de error de peso mínimo.

Como podemos observar en el primer ejemplo si la palabra sufre demasiados cambios, IMLD pensará que la palabra de código más cercana es la que se envió, así debemos encontrar una manera para que ésto no suceda. Además, en el segundo ejemplo podemos observar que existen códigos donde puede suceder que se requiera una retransmisión, todo esto porque los cambios que provocó el ruido es tal que coloca a la palabra recibida en un lugar equidistante a nuestras palabras de código.

Por lo anterior es necesario poder elegir correctamente nuestro código como vimos, es de **saber elegir** con propiedad nuestro código, ya que de eso dependerá la efectividad del mismo.

Otra cosa importante de remarcar es que diferentes códigos tendrán resultados distintos, como se ve en los ejemplos, por ello resulta necesario tener una herramienta que facilite poder comparar la efectividad entre códigos y a partir de eso elegir el más apropiado.

1.6.1. La confiabilidad que tiene nuestro método MLD

Primero debemos recordar que cuando definimos $\phi_p(\cdot,\cdot)$ se hacía un estudio estadístico de eventos de Bernoulli, y con ello definimos $\phi_p(\cdot,\cdot)$. Sí queremos estudiar que tan confiable es nuestro método MLD debemos ver los casos donde tengamos una palabra cerca a una palabra código, y verificar que tan probable es que esto ocurra. Así podemos aplicar la **función de probabilidad** de la **distribución binomial de Poisson**

$$P(K=k) = \sum_{A\in F_k} \prod_{i\in A} p_i \prod_{j\in A^c} (1-p_j)$$

donde F_k es el conjunto de todos los subconjuntos de k enteros que se pueden seleccionar de $\{1,2,3,...,n\}$. Así podremos tener la probabilidad de que MLD concluya correctamente.

Con lo anterior basta saber cuales son las palabras más cercanas a una de nuestro código y aplicar la distribución binomial de Poisson, la cual es aplicar ϕ, a cada una de las palabras, así tenemos una nueva definición:

Definición 1.6.5. *El conjunto de todas las palabras de longitud n, que están únicamente cerca a $v \in C$ (v fijo) de cualquier otra palabra de C, donde C es nuestro código, se denotará como $L(v)$.*

Entonces al aplicar la función de probabilidad a $L(v)$ (o sea $F_k = L(v)$), tendremos la siguiente definición.

Definición 1.6.6. *La probabilidad de que MLD concluya correctamente que una palabra v del código fue enviada, se calcula con la fórmula siguiente:*

$$\theta_p(C,v) = \sum_{w\in L(v)} \phi_p(w,v)$$

donde p es la probabilidad de que un dígito llegue correctamente en un canal BSC.

Observaciones:

- θ_p se puede usar para comparar códigos y poder ver cuál es mejor, dejando fijo una palabra v, y sean C_1 y C_2, códigos que contienen a v, entonces C_1 será mejor código para v, si $\theta_p(C_1, v) > \theta_p(C_2, v)$.
- $\theta_p(C, v)$ es la suma sobre las palabras w de $L(v)$ de las probabilidades de que los patrones de error $v + w$ ocurran durante la transmisión.
- Entre más cerca esté de 1 la probabilidad $\theta_p(C, v)$, entonces C será una buena elección de código para la palabra v.

Para finalizar esta sección observemos unos ejemplos donde apliquemos nuestra nueva fórmula.

Ejemplo 1.6.7. *Sea* $p = 0{,}9$, $|M| = 2, n = 3$, *y* $C = \{001, 101\}$, *encontrar la probabilidad de que IMLD concluya correctamente que* 001 *fue enviado.*

Solución:

Primero encontremos $L(001)$, pero para esto basta con retomar de la tabla que hicimos anteriormente y tendremos:

$$L(001) = \{000, 001, 010, 011\}$$

Finalmente usando la fórmula obtenida anteriormente tendremos:

$$\begin{aligned}\theta_p(C, 001) &= \phi_p(000, 001) + \phi_p(001, 001) + \phi_p(010, 001) + \phi_p(011, 001) \\ &= p^2(1-p) + p^3 + p(1-p)^2 + p^2(1-p) \\ &= 2p^2(1-p) + p^3 + p(1-p)^2 \\ &= 0{,}9.\end{aligned}$$

Por lo tanto la probabilidad de que IMLD concluya que 001 fue enviado es 0,9 que es bastante bueno.

Ejemplo 1.6.8. *Sea* $p = 0{,}9$, $|M| = 3, n = 3$ *y* $C' = \{000, 001, 110\}$, *encontrar la probabilidad de que IMLD concluye que fue enviado* 001.

Solución:

Procedemos como en el ejemplo anterior, además teniendo en cuenta que este código ya lo habíamos trabajado anteriormente, así $L(001) = \{001, 011, 101\}$, con esto tendremos:

$$\begin{aligned}\theta_p(C', 001) &= \phi_p(001, 001) + \phi_p(011, 001) + \phi_p(101, 001) \\ &= p^3 + p^2(1-p) + p^2(1-p) \\ &= 2p^2(1-p) + p^3 \\ &= 0{,}891.\end{aligned}$$

La probabilidad de que IMLD concluya que 001 fue enviado es de 0,891, de hecho esta probabilidad solo es aceptable, ya que es menos que 0,9, de haber tenido una probabilidad de un 0,8 o menos es una probabilidad bastante mala y la elección del código no fue muy buena.

De los dos últimos ejemplos examinamos una misma palabra 001 y obtuvimos $\theta_p(C, 001) = 0{,}9$ y $\theta_p(C', 001) = 0{,}891$, es claro que para el código C tenemos una probabilidad mayor entonces el código C es una mejor opción para enviar 001, que C' (C es más confiable que C').

Una conclusión bastante obvia es que todo esfuerzo por corregir y detectar errores es simplemente poder elegir un código que tenga una confianza sumamente aceptable para el IMLD, para ello enumeraremos tres simples pasos para hacer una sabia elección.

1. No debemos elegir palabras de longitud muy largas ya que estas tardarán más en transmitirse, por ende debemos elegir la longitud n, de tal manera que la tasa de información sea lo más cercano a 1 (lo que en otras palabras sería, no agregar mucha redundancia).

2. Debemos elegir C no tan grande, de tal manera que hacer un análisis como el descrito en esta sección no se lleve demasiado tiempo.

3. Debe elegirse C de tal manera que la probabilidad de MLD sea muy alta para cada palabra del código C, ya que que si la probabilidad es muy baja, esto significa que suceden muchos errores en la transmisión y como vimos IMLD termina concluyendo erróneamente que es otra palabra de código distinta a la que se envió realmente.

Es por esto que el objetivo principal de la teoría de código es **encontrar conjuntos C de palabras que cumplan los tres pasos anteriores**.

1.7. Códigos detectores de errores

Es necesario formalizar cuándo un código C es detector de errores. Para ello recordemos los que tenemos hasta el momento, sabemos que si recibimos una palabra w y esta

no pertenece a C, entonces sabemos que ha ocurrido un error en la transmisión, además hemos encontrado que para saber cuál palabra de código fue enviado hemos definido el patrón de error como $u = v + w$, para cada v palabra en C, y debido a que K^n es un espacio vectorial y al hacer todas las posibles sumas de $w \in K^n$ con cada $v \in C$, tendremos que toda palabra del espacio vectorial funciona como patrón de error.

Por lo tanto conocemos los patrones de error que pueden resultar en la transmisión y entonces el problema de que C detecte un error; que es $w \notin C$, se pasa a saber si C es capaz de detectar el patrón de error u, ya que sabemos todos los posibles patrones de error y haciendo un despeje podemos conocer la palabra w a recibir de ocurrir un error ($w = u + v$). Con esto tenemos la siguiente definición:

Definición 1.7.1. *El código C detectará un patrón de error u, si y solo si $u + v$ no es una palabra de código de C, para cada $v \in C$.*

Veamos un ejemplo para ver como funciona nuestra definición.

Ejemplo 1.7.2. *Sea $C = \{001, 101, 110\}$, encontrar cuándo C no detecta los patrones de error $011, 001$ y 000.*

Solución:

Para saber cuál de esos patrones de error puede detectar C, basta que hagamos todas las sumas con cada palabra de código y ver cuales no pertenecen al código.

- Para 011, tenemos $011 + 001 = 010 \notin C$, $011 + 101 = 110 \in C$ y $011 + 110 = 101 \in C$.
- Para 001, entonces $001 + 001 = 000 \notin C$, $001 + 101 = 100 \notin C$ y $001 + 110 = 111 \notin C$.
- Para 000, tendremos $000 + 001 = 001 \in C$, $000 + 101 = 101 \in C$ y $000 + 110 = 110 \in C$.

Por lo tanto solo el patrón de error 001 puede detectar C, de los tres que se pidieron examinar, los demás están descartados porque al menos una suma ($u + v = w$), pertenece al código C, y en la definición dice que debe cumplirse para toda suma que ninguna sea una palabra de código.

Hay ciertas cosas que podemos observar del ejemplo anterior, como cuando el patrón de error es la palabra cero, el código no lo puede detectar, además si se cumple que $v + w = 0$, para una palabra recibida w y una palabra del código v, entonces tampoco puede detectar el patrón cero, así tenemos las siguientes proposiciones:

Proposición 1.7.3. *Ningún código C, puede detectar el patrón de error cero ($u = \mathbf{0}$).*

Demostración. Sea C un código cualquiera, y sea el patrón de error cero u, entonces como es el patrón de error cero u cumple ser la palabra cero, y por definición es la identidad del espacio vectorial así tenemos:

$$\forall v \in C \Rightarrow u + v = v \in C$$

Lo cual hace que no pueda cumplir la definición, ya que para que C pueda detectar a u, la suma con cada v no debe quedar en C y por lo tanto tenemos que C no detecta el patrón de error cero (C se tomó de forma general, así que tenemos la prueba finalizada). □

Proposición 1.7.4. *Sea C un código, y además sea $v \in C$ y w una palabra recibida tal que $v + w = \mathbf{0}$, entonces $\mathbf{0}$ no puede ser detectado por C.*

Demostración. Ya que v y w, cumplen $v + w = \mathbf{0}$, entonces por propiedades de K^n, tenemos $v = w \Rightarrow v + \mathbf{0} = w$ y por hipótesis $v \in C \Rightarrow v + \mathbf{0} = w \in C$, entonces hemos encontrado una suma con el patrón de error $\mathbf{0}$ que resulta estar en el código entonces C no puede detectar el patrón $\mathbf{0}$. □

Y de hecho más aún, sucede algo interesante si C contiene la palabra cero.

Proposición 1.7.5. *Sea C un código que contiene la palabra cero como palabra código. Si el patrón de error u es una palabra de código, entonces C no puede detectar u.*

Demostración. Para demostrar esto procedamos por contradicción asumamos que C si detecta a u, pero si lo detecta cumple por definición:

$$\forall v \in C \Rightarrow u + v \notin C$$

en particular se debe cumplir para $v = \mathbf{0}$, entonces tenemos $\mathbf{0} + u \notin C \Rightarrow u \notin C$, pero esto es una contradicción ya que por hipótesis tenemos que $u \in C$. La contradicción viene de suponer que C podía detectar u, por lo tanto C no puede detectar u. □

Usando la tabla IMLD que construimos antes podemos saber cuales patrones de error el código es capaz de detectar, ya podemos interpretar la primera columna como todos los posibles patrones de error que nos pueden aparecer y las columnas de los patrones de error como las posibles palabras que podemos recibir. Si dada una fila ninguna de las palabras recibidas para un patrón de error es una palabra de código entonces C detectará dicho patrón.

Así tenemos el siguiente ejemplo:

Ejemplo 1.7.6. *Sea $|M| = 2, n = 3$, y $C = \{001, 101\}$, encontrar los patrones de error que detecta C.*

Solución

Patrón de error u	palabras recibidas w	
	$001+u$	$101+u$
000	001*	101*
001	000	100
010	011	111
011	010	110
100	101*	001*
101	100	000
110	111	011
111	110	010

Hemos colocado un asterisco, cuando la palabra $w = u + v$ pertenezca a C, como vemos en los casos $000, 100$, el código C no puede detectar estos patrones, en el resto si es posible que lo detecte.

Esta verificación manual para saber cuales patrones de error puede detectar el código C, es poco práctica ya que si el código es muy grande y el espacio vectorial también tomaría mucho tiempo para poder encontrar una forma más fácil de poder verificar que un patrón de error es detectable o no. Primero introduciremos la siguiente definición:

Definición 1.7.7. *La **distancia del código** C es la distancia mínima entre todas las palabras de código, tomadas de dos en dos y sin que sean iguales las palabras; lo que significa:*

$$d = \text{mín}\{d(v, w) : v, w \in C, v \neq w\}.$$

Ya con esto podemos enunciar un teorema que será de gran utilidad para saber cuando un código puede detectar cierto tipo de patrones de error.

Teorema 1.7.8. *Un código C de distancia d detectará al menos todos los patrones de error distintos de cero de peso menor o igual a $d-1$. Además, hay al menos un patrón de error de peso d que C no detectará.*

Demostración. Sea u un patrón de error distinto de cero con $wt(u) \leq d-1$, y tomemos un v arbitrario en C, así tenemos:

$$d(v, v+u) = wt(v+v+u) = wt(u) \leq d-1 < d.$$

Como C tiene distancia d, $v+u$ no está en C. Por lo tanto, C detecta a u.

De la definición de d, existen dos palabras de código v y w tal que $d(v, w) = d$. Ahora consideremos el patrón de error $u = v + w$, de lo anterior al despejar w, tenemos $w = v + u$ y como $w \in C$, entonces $v + u \in C$, por lo que C no detectará el patrón de error u de peso d. □

Observaciones:

1. El código C, puede llegar a detectar algunos o todos los patrones de error de peso mayor a d, pero no todos los de peso d.

   Por ejemplo el caso del ejemplo 1.7.6 el código es $C = \{001, 101\}$, la distancia d de C, es 1, y como vemos en la tabla siguiente :

Patrón de error u	palabras recibidas w	
	$001 + u$	$101 + u$
000	001*	101*
001	000	100
010	011	111
011	010	110
100	101*	001*
101	100	000
110	111	011
111	110	010

   No detecta el patrón 100, y este tiene peso 1, lo cual concuerda con el teorema ya que la distancia de C también es 1.

   En el caso de los demás patrones distinto de cero, tienen peso mayor que uno y los detecta a todos; esto último no sucede siempre.

2. Con lo discutido anteriormente, queremos hacer énfasis en que el teorema no restringe a que un código C, no pueda detectar patrones de error de peso mayor a la distancia del código, puede y generalmente sucederá que C detectará algunos de mayor peso.

Para finalizar esta sección damos una última definición que tiene que ver con los patrones de error que puede detectar el código.

Definición 1.7.9. *Un código C se llamará t-**detector de errores**, si detecta todos los patrones de error de peso a lo sumo t y no detecta al menos un patrón de error de peso $t + 1$.*

Debido a la definición anterior, y usando el teorema 1.7.8, un código C de distancia d, es un $(d - 1)$-detector de errores.

1.8. Códigos correctores de errores

Algo más importante que detectar errores, es poder corregir el error, la técnica que tenemos hasta el momento para poder corregir un error es brindado por IMLD, la cual nos dice:

Si se recibe w y solo existe una palabra v del código que está más cerca que cualquier otra palabra del código, entonces la palabra que fue enviada fue v.

Esto ofrece una manera de corregir los errores pero al igual que detectar, estar corroborando mediante procesos mecánicos es demasiado engorroso cuando el código es muy grande, por ello debemos idear una forma de poder predecir o al menos saber si cierto tipos de palabras w se pueden corregir.

Al igual que en la sección anterior en vez de estudiar el comportamiento de la palabra recibida, estudiemos el patrón de error, así si enviamos una palabra v de código y recibimos w, entonces tenemos el patrón de error $u = v + w$, y si IMLD concluye correctamente que se envió v, y si esto sucede cada vez que u ocurre, podemos decir que C corrige el patrón de error u, con esto podemos dar la siguiente definición:

Definición 1.8.1. *Un código C corrige el patrón de error u, si para todo $v \in C$, $u + v$ es la palabra más cercana a v, que cualquier otra palabra de código en C.*

Veamos a continuación algunos ejemplos:

Ejemplo 1.8.2. *Sea $|M| = 2, n = 3$, y $C = \{001, 101\}$, ¿C puede corregir el patrón de error* 011*?*

Solución

Usando la definición anterior tenemos que hacer el análisis para cada uno de las palabras de código y verificar que siempre $v + u$ es la palabra más cercana a v.
Primero veamos para $v = 001$:

$$d(001, u + 001) = d(001, 010) = 2$$
$$d(101, u + 001) = d(101, 010) = 3$$

Por último veamos para $v = 101$:

$$d(001, u + 101) = d(001, 110) = 3$$
$$d(101, u + 101) = d(101, 110) = 2$$

Como vemos solo v es el más próximo a $u + v$, por lo tanto C corrige 011.

Ejemplo 1.8.3. *Sea $|M| = 3, n = 3$ y $C = \{000, 001, 110\}$, ¿C puede corregir $u = 010$?*

Solución

De igual manera que en el ejercicio anterior analizamos todas las posibilidades para v.
Para $v = 001$

$$d(000, u + 110) = d(000, 011) = 2$$
$$d(001, u + 110) = d(001, 011) = 1$$
$$d(110, u + 110) = d(110, 011) = 2$$

v es el más cercano

Para $v = 000$

$$d(000, u + 000) = d(000, 010) = 1$$
$$d(001, u + 000) = d(001, 010) = 2$$
$$d(110, u + 000) = d(110, 010) = 1$$

En este caso tanto v como 110 están a la misma distancia mínima de w, por ende no cumple la definición, ya que debe verificarse para todos, y hemos encontrado donde no cumple, por lo tanto C no puede corregir $u = 010$.

Podemos utilizar la tabla que construimos para IMLD, de tal manera que nos ayude a poder encontrar qué patrones de errores el código C puede detectar como hemos visto en los ejemplos anteriores el patrón de error debe funcionar para todos los elementos del código, así pues en las columnas de la tabla de IMLD donde encontramos los patrones de error, si un patrón de error aparece en todas estas columnas marcado con un asterisco, entonces el código puede corregirlo en caso contrario se concluye que C no puede corregirlo.

Recuerde que solo analizamos cuando la palabra $u + v$ es la más cercana a v que cualquier otra palabra en el código, en los casos donde IMLD requiere retransmisión se tomarán como si no estuvieran marcados.

Así los ejemplos anteriores podemos volver a resolverlos pero ahora haciendo uso de la tabla de IMLD.

Ejemplo 1.8.4. *Sea* $|M| = 2, n = 3$, *y* $C = \{001, 101\}$, *¿C puede corregir el patrón de error* 011*?*

Recibido w	Patrón de error $001 + w$	$101 + w$	Descodificación v
000	001*	101	001
001	000*	100	001
010	011*	111	001
011	010*	110	001
100	101	001*	101
101	100	000*	101
110	111	011*	101
111	110	010*	101

Como podemos ver en 011 aparece marcado en las dos columnas, por tanto C lo puede corregir.

Ejemplo 1.8.5. *Sea* $|M| = 3, n = 3$ *y* $C = \{000, 001, 110\}$, *¿C puede corregir* $u = 001$*?*

Recibido	Patrón de error			Descodificación
w	$000+w$	$001+w$	$110+w$	v
000	000*	001	110	000
001	001	000*	111	001
010	010*	011	100*	-
011	011	010*	101	001
100	100*	101	010*	-
101	101	100*	011	001
110	110	111	000*	110
111	111	110	001*	110

En este caso para 001, solo aparece marcado en la tercera columna, mientras que en la primera y segunda columna no está marcado, por ende podemos concluir que C no corrige 001.

De igual forma, si un patrón de error aparece marcado en todas las columnas, y además al menos una de las filas donde está marcado requiere retransmisión, entonces se considerará como no válida.

Al igual en el caso de saber si un código puede detectar errores, que los procesos largos son poco prácticos, es necesario poder saber de manera más práctica, cuándo un código es corrector de errores. Y para ello, enunciaremos el último teorema importante de este capítulo.

Teorema 1.8.6. *Un código C de distancia d corregirá todos los patrones de error de peso menor o igual que $\lfloor (d-1)/2 \rfloor$. Además, hay al menos un patrón de error de peso $1+\lfloor (d-1)/2 \rfloor$ que C no corregirá.*

Nota: Recuerde que $\lfloor x \rfloor$, es la función menor entero o función piso.

Demostración. Sea u un patrón de error de peso $wt(u) \leq (d-1)/2$. Sean $v \in C$ fijo y w una palabras de código en C arbitraria tal que $w \neq v$.

Queremos mostrar que $d(v, v+u) < d(w, v+u)$ (note que aquí v se mantiene pero w recorre al resto de palabras de C).

$$
\begin{aligned}
d(w, v+u) + d(v+u, v) &\geq d(w, v) \\
d(w, v+u) + d(v+u, v) &\geq d \quad \text{Porque } d(w, v) \geq d\text{; definición de distancia de } C, \\
d(w, v+u) + wt(u) &\geq 2wt(u) + 1 \\
d(w, v+u) &\geq wt(u) + 1 \\
d(w, v+u) &\geq d(v, v+u) + 1 > d(v, v+u) \\
\therefore d(v, v+u) &< d(w, v+u)
\end{aligned}
$$

Ya que $wt(u) = d(v+u, v)$, y $2wt(u) + 1 \leq d$.

Y como v era cualquiera, se cumple para el resto de palabras de C lo anterior. Por lo tanto, C corrige u.

Para la otra parte del teorema, observemos que d es la distancia del código C y por tanto existen v y w palabras de código tal que $d(v, w) = d$. Formamos un patrón de error u cambiando $d - 1 - \lfloor (d-1)/2 \rfloor$ de los d unos en $v + w$ a ceros. Entonces

$$
\begin{aligned}
d(v, v+u) &= wt(u) = 1 + \lfloor (d-1)/2 \rfloor \\
d(w, v+u) &= wt(w+v+u) = d(v+w, u) \\
&= d - (1 + \lfloor (d-1)/2 \rfloor).
\end{aligned}
$$

Como d es un entero positivo analicemos por casos:

i Si d es impar, es decir $d = 2t + 1$, entonces

$$
\begin{aligned}
d(v, v+u) &= wt(u) = 1 + (2t)/2 = 1 + t \\
d(w, v+u) &= 2t + 1 - (1 + t) = t.
\end{aligned}
$$

entonces $d(v, v+u) > d(w, v+u)$.

ii Si d es par, $d = 2t$, entonces

$$
\begin{aligned}
d(v, v+u) &= 1 + \lfloor t - 1/2 \rfloor = t \\
d(w, v+u) &= 2t - t = t.
\end{aligned}
$$

En cualquier caso, $d(v, v+u) \geq d(w, v+u)$, entonces $v + u$ no está más cerca de v que de la palabra de código w. Por lo tanto, C no corrige el patrón de error u.

□

Observación:

1. El teorema 1.8.6 no impide que un código C, de distancia d, corrija los patrones de error de peso mayor que $\lfloor (d-1)/2 \rfloor$.
2. En este teorema a diferencia del teorema 1.7.8, sí da información de patrones de error cero.

Ahora veamos algunos ejemplos:

Ejemplo 1.8.7. *Sea* $|M| = 2, n = 3$, *y* $C = \{001, 101\}$, *¿C puede corregir el patrón de error* 011*?*

Solución

En este caso C tiene distancia $d = 1$, y si observamos el patrón de error 011, tiene peso 2, por lo que el teorema no garantiza nada sobre este caso, es necesario examinar la tabla IMLD, la cual presentamos a continuación pero solo en las filas donde aparece el patrón de error 011

Recibido	Patrón de error		Descodificación
w	$001 + w$	$101 + w$	v
010	011*	111	001
110	111	011*	101

Como vemos aparece en todas las columnas, por tanto C corrige 011, como vemos este es un ejemplo donde C detecta un patrón de error mayor que $\lfloor (d-1)/2 \rfloor$.

Además todo patrón de error $\lfloor (d-1)/2 \rfloor$, lo detecta, como $d = 1$, buscamos patrones de error menores o igual a cero, pero el único que tiene peso cero es la palabra cero, lo cual concuerda pues 000 era un patrón de error que sí puede corregir como se ve en la tabla del ejemplo 1.8.5

Ejemplo 1.8.8. *Sea* $|M| = 3, n = 3$ *y* $C = \{000, 001, 110\}$, *¿C puede corregir* $u = 010$*?*

Solución

Este código tiene también distancia uno, y u tiene peso 1, para verificar que este patrón de error se corrige o no, veamos de nuevo la tabla IMLD, así como en el ejemplo anterior.

Recibido	Patrón de error			Descodificación
w	$000+w$	$001+w$	$110+w$	v
010	010*	011	100*	-
011	011	010*	101	001
100	100*	101	010*	-

Como vemos 010, no puede ser corregido por C, por tanto este patrón de error no se puede detectar, y es el que hace mención el teorema 1.8.6, ya que como $d = 1 \Rightarrow 1 + \lfloor (d-1)/2 \rfloor = 1$ y en efecto el peso de 010 es uno también.

Para finalizar veamos algunas proposiciones que son consecuencia de lo definido y demostrado antes:

Proposición 1.8.9. *El patrón de error cero, siempre es corregido.*

Demostración. Sea C un código cualquiera de palabras de longitud n, y sea u el patrón de error cero, tomando $v \in C$ fijo, y w arbitrario también en C, tal que $v \neq w$, tenemos:

$$d(v, u+v) = wt(u+v+v) = wt(u) = wt(\mathbf{0}) = 0$$
$$d(w, u+v) = wt(w+u+v) = wt(w+0+v) = wt(w,v) = d(w,v)$$

y como $w \neq v \Rightarrow d(w,v) > 0 \Rightarrow d(w, u+v) > d(v, u+v)$, por lo tanto C corrige a u, y como v era cualquiera, se cumple lo anterior para todas las palabras de C, así C corrige el patrón de error cero. □

Definición 1.8.10. *Un código C se dice que es **t-corrector de errores**, si corrige todos los patrones de error de peso a lo sumo t y no corrige al menos un patrón de error de peso $t+1$.*

Por el teorema 1.8.6, un código de distancia d, es un código $\lfloor (d-1)/2 \rfloor$-corrector de errores.

Hemos aprendido que nuestro conjunto de códigos, tiene la habilidad de detectar y corregir errores que pueden ocurrir durante la transmisión, además adoptamos un mecanismo que nos ayuda a elegir la palabra más probable de haber sido enviada. Lo que queda es un trabajo de poder ir explorando las propiedades de K^n y de nuestros códigos C, para formar un mecanismo que nos permita no solo saber cuál palabra es la más probable de haber sido enviada, sino también, en qué dígito ocurre el error o en qué dígitos, todo esto y aún más lo veremos en la investigación.

Capítulo 2

Códigos lineales

Los códigos lineales son códigos correctores de errores con la característica que la suma de dos o más palabras del código siguen siendo una palabras de código. Estos códigos son muy útiles debido a la versatilidad de una codificación y decodificación eficiente. De hecho a lo largo de este capítulo estudiaremos códigos lineales de bloque (como se estableció en el capítulo 1), aunque existen dos tipos más, concentraremos nuestra atención a los código de bloques. Al estudiar este tipo de código se desarrollará una manera más fácil para detectar y corregir errores. Logrando así establecer la base de la corrección de errores para los códigos lineales y con ello dotar al código lineal de una base que lo genere y de establecerle un dual.

2.1. Códigos lineales

En esta sección introduciremos una clase especial de código que nos permitirá introducir una mayor estructura matemática a nuestros códigos, de hecho el resto de códigos que se expondrán son de éste tipo. La razón por la cual son importantes estos códigos es porque permiten cálculos más fáciles para algunos de los procesos expuestos en el capítulo 1.

Definición 2.1.1. *Un código C es llamado **código lineal**, si $\forall u, v \in C$ la palabra $u + v$ es una palabra del código de C.*

Ejemplo 2.1.2. *Sea $C = \{000, 001, 010, 011\}$ es un código lineal.*

C es un código lineal ya que todas las sumas de dos palabras de código son otra palabra de código, esto es:

$000 + 001 = 001$ $001 + 010 = 011$ $010 + 011 = 001$

$000 + 010 = 010$ $001 + 011 = 010$ $011 + 001 = 010$

$000 + 011 = 011$ $010 + 001 = 011$ $011 + 010 = 001$

$001 + 001 = 000$ $010 + 010 = 000$ $011 + 011 = 000$

Como vemos todas las sumas resultan ser una palabra de C, ésta característica de los códigos lineales se conoce como: C es cerrado bajo la suma.

También es necesario mencionar que de acuerdo a la definición de código lineal la palabra cero siempre está en los códigos lineales; esto es así porque en la sección 1.4 se demostró que cada palabra es su mismo inverso bajo la suma así para una palabra v tenemos $v + v = \mathbf{0}$, pero si C es un código lineal, tenemos:

$$v + v \in C \Rightarrow \mathbf{0} \in C$$

La condición anterior de que la palabra cero esté en el código lineal C es solo necesaria ya que se puede tener un código que tenga la palabra cero pero no sea un código lineal. Por ejemplo:

Ejemplo 2.1.3. *Sea* $C = \{000, 001, 010\}$ *no es un código lineal.*

El código anterior no es un código lineal ya que $001 + 010$ no es una palabra de C, y además este código tiene a la palabra cero. Como vemos el hecho que un código tenga la palabra cero no garantiza que sea lineal, lo que si podemos decir es que si el código **no** tiene la palabra cero entonces C **no** es un código lineal.

Observaciones: De lo discutido anteriormente y de la definición 2.1.1, podemos concluir:

1. Un código lineal C es un subespacio vectorial en K^n; esto es así porque cada palabra de C, tiene sus inversos en C (ya que son ellos mismos), la palabra cero está en C (porque C es un código lineal), es cerrado bajo el producto por escalar (trivial ya que nuestro campo solo tiene dos elementos cero y uno, y cero provoca la palabra cero para cualquier palabra y uno las mismas palabras), y finalmente por definición C es cerrado bajo la suma.

2. En general para el espacio vectorial K^n, un subconjunto A no vacío es subespacio si y solo si es cerrado bajo la suma (por lo expuesto antes es claro que la condición de que sea cerrado bajo el producto por escalar es consecuencia de que sea cerrado bajo la suma).

Al inicio de esta sección se habla de la importancia de estos códigos en el sentido que permite hacer cálculos más fácilmente, para demostrar esto veamos la definición de distancia de un código aplicada a un código lineal.

$$\begin{aligned} d &= \text{mín}\{d(u,v) : u \neq v \wedge u, v \in C\} \\ &= \text{mín}\{wt(u+v) : u \neq v \wedge u, v \in C\} \\ &= \text{mín}\{wt(w) : w \neq \mathbf{0} \wedge w \in C\}. \end{aligned}$$

De lo anterior tenemos:

Definición 2.1.4. *La **distancia** de un código lineal C es igual al peso mínimo de alguna palabra de código distinta de la palabra cero.*

Como vemos la facilidad de cálculos que nos permiten los códigos lineales es sumamente grande y por ello en las secciones restantes seguiremos explotando sus propiedades como subespacio vectorial para mejorar las técnicas de codificación y decodificación.

2.2. El código dual de C

En esta sección se presentará un tipo de código lineal muy especial. La importancia de éste código lineal radica en que permitirá mejorar nuestras herramientas de codificación y decodificación en las siguientes secciones. Para ello iremos recordando algunos resultados del álgebra lineal que iremos reescribiendo para el caso particular de nuestro espacio vectorial K^n.

Primero recordemos que si tenemos v un vector, decimos que es combinación lineal de los vectores $\{u_1, \ldots, u_k\}$ si existen escalares $a_1, \ldots, a_k$ tal que:

$$v = a_1u_1 + \ldots + a_ku_k.$$

Además al conjunto de todas las combinaciones lineales de $S = \{u_1, \ldots, u_k\}$ es un subespacio vectorial y se denota por $< S >$. Al conjunto $< S >$ se le llama **span de S** o subespacio generado de S (también del álgebra lineal se sabe que $< S >$ es el subespacio más pequeño que contiene a S).

Debemos notar que si $S \subseteq K^n$ por definición de $< S >$ cumple la definición 2.1.1 y por lo tanto es un código lineal, además cuando S cumple ser un código lineal se tiene $< S >= S$ ($C =< C >$); ya que S sería el subespacio más pequeño que contiene a S.

De lo anterior discutido vemos que todo código lineal es generado por un subconjunto de palabras de K^n. Además podemos enunciar el siguiente teorema:

Teorema 2.2.1. *Para cada subconjunto S de K^n, el código $C =< S >$ generado por S consiste precisamente de las siguientes palabras: la palabra cero, todas las palabras de S y todas las sumas de dos o más palabras en S.*

Demostración. Sea $S \subseteq K^n$ tal que $S = \{u_1, \ldots, u_k\}$, así por definición de $< S >$ tenemos:

$$\forall v \in < S > \Rightarrow v = a_1u_1 + \ldots + a_ku_k, \text{ donde } a_1, \ldots, a_k \in K$$

como K solo tiene dos elementos cero y uno, se tiene que los a_i cumplirán para cada combinación lineal lo siguiente:

1. Existe una combinación lineal donde todos los a_i son cero y así resulta la palabra cero.
2. Existe un combinación lineal donde solo se tome un a_i igual a uno y el resto iguales a cero, así tendremos todas las palabras de S, además pueden existir combinaciones lineales donde más de uno escalar es igual a uno. De esta manera tendremos palabras que serán la suma de dos o más palabras de S.

□

Ejemplo 2.2.2. *Sea $S = \{010, 011, 111\}$. Encontrar el código $C =< S >$ generado por S.*

Solución

Para encontrar el código lineal solo hacemos uso del teorema 2.2.1, así tendremos primero las palabras de S y la palabra cero, lo cual genera el siguiente conjunto de palabras: $\{000, 010, 011, 111\}$, ahora agregamos la suma de dos palabras de S al conjunto anterior, así tendremos el siguiente conjunto:

$$\{000, 010, 011, 111, 010 + 011, 010 + 111, 011 + 111\} = \{000, 010, 011, 111, 001, 101, 100\}$$

Finalmente agregamos la suma de las tres palabras de S al conjunto construido anteriormente:

$$\{000, 010, 011, 111, 001, 101, 100, 010 + 011 + 111\} = \{000, 010, 011, 111, 001, 101, 100, 110\}$$

Por lo tanto el código que genera S es $C =< S >= \{000, 010, 011, 111, 001, 101, 100, 110\}$.

Podemos definir de manera natural a K^n un producto escalar (Como se hace en $\mathbb{R}^n$). Lo que significa que el producto escalar por definir será la suma de los productos de las coordenadas de dos palabras, dicha suma y productos se realizan en K, así tenemos:

Definición 2.2.3. *Sea $v = (a_1, \ldots, a_n)$ y $w = (b_1, \ldots, b_n)$ palabras de K^n. Definimos el **producto escalar** o **producto punto** $v \cdot w$ de v y w como:*

$$v \cdot w = a_1b_1 + \ldots + a_nb_n.$$

Obsérvese que los productos $a_i b_i$ y la suma de ellos se hacen en el campo K, por ende todas las operaciones se hacen módulo 2.

Definido así el producto escalar para nuestro espacio vectorial K^n cumple las siguientes propiedades:

Proposición 2.2.4. *Para $u, v, w \in K^n$ y $a \in K$, el producto escalar definido en 2.2.3. cumple:*

1. $u \cdot v = v \cdot u$
2. $u \cdot (v + w) = u \cdot v + u \cdot w$
3. $a(v \cdot w) = (av) \cdot w = v \cdot (aw)$
4. $u \cdot v \geqslant 0$

Demostración. Sean $u, v, w \in K^n$ y $a \in K$ tal que $u = u_1 u_2 \dots u_n$, $v = v_1 v_2 \dots v_n$ y $w = w_1 w_2 \dots w_n$, donde $u_i, v_i, w_i \in K, \forall i = 1, \dots, n$, así:

(1) $u \cdot v = v \cdot u$

Haciendo uso de las propiedades del campo K tendremos:

$$\begin{aligned} u \cdot v &= (u_1 u_2 \dots u_n) \cdot (v_1 v_2 \dots v_n) \\ &= u_1 v_1 + u_2 v_2 + \dots + u_n v_n \\ &= v_1 u_1 + v_2 u_2 + \dots + v_n u_n \\ &= (v_1 v_2 \dots v_n).(u_1 u_2 \dots u_n) \\ &= v \cdot u \end{aligned}$$

(2) $u \cdot (v + w) = u \cdot v + u \cdot w$

$$\begin{aligned} u \cdot (v + w) &= (u_1 u_2 \dots u_n) \cdot (v_1 v_2 \dots v_n + w_1 w_2 \dots w_n) \\ &= (u_1 u_2 \dots u_n) \cdot ((v_1 + w_1)(v_2 + w_2) \dots (v_n + w_n)) \\ &= (u_1(v_1 + w_1)) + (u_2(v_2 + w_2)) + \dots + (u_n(v_n + w_n)) \\ &= u_1 v_1 + u_1 w_1 + u_2 v_2 + u_2 w_2 + \dots + u_n v_n + u_n w_n \\ &= u_1 v_1 + u_2 v_2 + \dots + u_n v_n + u_1 w_1 + u_2 w_2 + \dots + u_n w_n \\ &= u \cdot v + u \cdot w \end{aligned}$$

(3) $a(v \cdot w) = (av) \cdot w = v \cdot (aw)$

Primero analicemos la igualdad: $a(v \cdot w) = (av) \cdot w$

$$\begin{aligned} a(v \cdot w) &= a(v_1 v_2 \dots v_n) \cdot (w_1 w_2 \dots w_n) \\ &= a(v_1 w_1 + v_2 w_2 + \dots + v_n w_n) \\ &= av_1 w_1 + av_2 w_2 + \dots + av_n w_n \\ &= (av_1) w_1 + (av_2) w_2 + \dots + (av_n) w_n \\ &= (av) \cdot w \end{aligned}$$

Ahora realizando los mismos pasos para: $a(v \cdot w) = v \cdot (aw)$

$$\begin{aligned} a(v \cdot w) &= a(v_1 v_2 \dots v_n) \cdot (w_1 w_2 \dots w_n) \\ &= a(v_1 w_1 + v_2 w_2 + \dots + v_n w_n) \\ &= av_1 w_1 + av_2 w_2 + \dots + av_n w_n \\ &= (av_1) w_1 + (av_2) w_2 + \dots + (av_n) w_n \\ &= v_1 (aw_1) + v_2 (aw_2) + \dots + v_n (aw_n) \\ &= v \cdot (aw) \end{aligned}$$

Al juntar las dos igualdades demostradas anteriormente tenemos: $a(v \cdot w) = (av) \cdot w = v \cdot (aw)$.

(4) $u \cdot v \geqslant 0$

Esta igualdad es cierta pues de la definición de producto escalar tenemos que al hacer la suma y producto módulo 2 las únicas dos posibilidades son cero y uno, sin importar que tipo de palabras le apliquemos la definición 2.2.3.

□

Observación: De la definición 2.2.3. tenemos que no solo la palabra cero cumple $u \cdot u = 0$, sino de hecho toda palabra que tenga un número par de unos en sus dígitos, por ejemplo si tenemos en K^6, la palabra $v = 011101$, el producto escalar de v es:

$$v \cdot v = 0 + 1 + 1 + 1 + 0 + 1 = 0 \bmod(2).$$

Esto quiere decir que nuestra definición de "producto escalar" es realmente un **semi-producto escalar** o **semi-producto interno**, ya que para ostentar el nombre de **producto escalar** o **producto interno** debe cumplir que solo la palabra cero tiene como resultado cero en su producto interno (solo ésta parte falla pues de acuerdo a la proposición 2.2.4. las demás propiedades las cumple). A pesar de todo lo anterior a lo largo del trabajo omitiremos la palabra **semi**. Sin embargo se debe mantener en cuenta que no es un producto escalar en toda regla.

Definición 2.2.5. *Dos palabras v, u en K^n se dice que son **ortogonales**, si $v \cdot u = 0$.*

Ejemplo 2.2.6. *Consideremos $v = 0110$ y $u = 1000$ en K^4, así tenemos:*

$$v \cdot u = (0110) \cdot (1000) = 0.1 + 1.0 + 1.0 + 0.0 = 0 + 0 + 0 + 0 = 0$$

Por lo tanto v y u son ortogonales.

Proposición 2.2.7. *Toda palabra de peso par en K^n es ortogonal a sí mismo.*

Demostración. Sea $v \in K^n$ tal que $wt(v) = 2t$ donde t es un entero no negativo.

Esto quiere decir que la palabra v tiene una cantidad par de unos, así cuando realice el producto punto con sí mismo tendremos una suma par de unos, que modulo 2 es cero, lo que significa:

$$v \cdot v = 2t = 0$$

Por lo tanto es ortogonal a sí mismo. □

Con éste producto escalar vamos a definir a continuación un conjunto que será de mucha importancia en nuestro objetivo de corregir las palabras que enviamos.

Definición 2.2.8. *Sea S un subconjunto de K^n. Diremos que la palabra v de K^n es **ortogonal** a S, si $v \cdot u = 0$ para todo u en S.*

Definición 2.2.9. *Sea S subconjunto de K^n, al conjunto de todas las palabras ortogonales a S se denota como $S^\perp$ y se le llama **complemento ortogonal** a S.*

Con estas dos definiciones conozcamos el conjunto más importante de ésta sección.

Proposición 2.2.10. *Sea S un subconjunto de K^n y C un código lineal de K^n. Si $C =< S >$ entonces $C^\perp = S^\perp$ y se le llama a $C^\perp$ **código dual** de C.*

Demostración. Demostremos esto por doble inclusión. Sea $S = \{w_1, w_2, \ldots, w_m\}$, Así

$$\text{sea } v \in C^\perp \Longrightarrow v \cdot u = 0, \forall u \in C.$$

Como $C =< S >$ y por definición tenemos $S \subseteq C$, esto quiere decir si tomamos $w \in S$ en general por la inclusión $w \in C$, con esto tendremos que $v \cdot w = 0$ para todo w en S y por lo tanto $v \in S^\perp \Rightarrow C^\perp \subseteq S^\perp$.

Por otro lado sea $v \in S^\perp$, y por otro lado ya que $C =< S >$ todo $u \in C$ es una combinación lineal de cada palabra en S, con esto tenemos existen $a_1, \ldots, a_n$ en K tal que:

$$u = a_1 w_1 + \cdots + a_n w_n.$$

Ahora aplicamos producto escalar con v a u,

$$v \cdot u = v \cdot (a_1 w_1 + \cdots + a_n w_n) = a_1(v \cdot w_1) + \cdots + a_n(v \cdot w_n) = 0 + \cdots + 0 = 0.$$

Así tenemos que $v \cdot u = 0$ para todo u de C, tenemos con esto que $v \in C^\perp \Rightarrow S^\perp \subseteq C^\perp$ y por lo tanto $C^\perp = S^\perp$. □

Observación: El complemento ortogonal definido anteriormente y así como el dual son definidos también en todo espacio vectorial con producto escalar, así nuestro código dual cumplirá de igual forma muchas propiedades que se estudian en los espacios vectoriales en general, como por ejemplo que el dual del dual de un conjunto es el conjunto ($(C^{\perp})^{\perp} = C$), por el momento no necesitamos mayor información de este conjunto así que nos conformamos con lo demostrado en la proposición 2.2.10.

Veamos un ejemplo donde encontremos el código dual.

Ejemplo 2.2.11. *Encontrar el código dual $C^{\perp}$ de $C =< S >$, donde $S = \{0101, 1010, 1100\}$.*

Solución

Para encontrar el código dual de C, solo debemos encontrar el complemento ortogonal de S por la proposición 2.2.10. Así para encontrar el complemento ortogonal de S tomamos una palabra en general de K^4, $u = xyzw$ y hacemos el producto punto con cada palabra de S y luego resolvemos el sistema.

$$\begin{aligned} 0101 \cdot u &= 0 \\ 1010 \cdot u &= 0 \\ 1100 \cdot u &= 0 \end{aligned}$$

Así tendremos el siguiente sistema lineal:

$$\begin{aligned} y + w = 0 &\Rightarrow y = w \\ x + z = 0 &\Rightarrow x = z \\ x + y = 0 &\Rightarrow x = y \end{aligned}$$

De donde se deduce $x = y = z = w$, y como nuestro campo solo tiene dos opciones (cero y uno) esto quiere decir que las palabras solución son 1111 y 0000.

Por lo tanto el código dual de C es:

$$C^{\perp} = S^{\perp} = \{0000, 1111\}$$

Es necesario mencionar que usando la proposición 2.2.7. y la definición 2.2.8. existen códigos que están contenidos en su dual por ejemplo el código $C = \{000, 110\}$, cuyo código dual es $\{000, 001, 110, 111\}$, y como vemos $C \subseteq C^{\perp}$; esto es algo que en los espacios vectoriales en general con producto escalar no se cumple.

Recordemos que K^n es un espacio vectorial y por ende muchas de las definiciones y teoremas que se desarrollan en el álgebra lineal se siguen cumpliendo aquí por ejemplo linealmente independiente y linealmente dependiente, bases para el espacio vectorial, bases para los subespacios vectoriales, entre muchos resultados más. A continuación presentaremos los más relevantes para nuestra temática reescritos en particular para K^n.

Definición 2.2.12. *Un conjunto de palabras distintas de la palabra cero $S = \{v_1, v_2, \ldots, v_k\}$ es linealmente dependiente, si existen escalares (dígitos) $a_1, a_2, \ldots, a_n$ no todos iguales a 0 tal que:*

$$a_1v_1 + a_2v_2 + \cdots + a_nv_n = \boldsymbol{0}$$

En el caso contrario las palabras de S son linealmente independiente.

Ejemplo 2.2.13. *Las palabras 101 y 010 son linealmente independiente ya que:*

$$a(101) + b(010) = 000$$

De lo anterior tenemos $a = 0$ y $b = 0$. Por lo tanto se verifica que son linealmente independiente, esto no se cumple para las palabras 110, 010, 100 y 001, ya que se puede hacer la siguiente combinación lineal:

$$1(110) + 1(010) + 1(100) + 0(001) = 000$$

Ya que la combinación lineal se cumple usando los primeros tres escalares iguales a uno podemos concluir que son linealmente dependiente.

Definición 2.2.14. *Un subconjunto B de K^n es base de K^n, si:*

1. $K^n =< B >$.
2. *B es un conjunto de palabras linealmente independientes.*

*Llamaremos **palabra base** a cada palabra que esté en la base B.*

Observación: De la definición anterior podemos concluir:

1. La base B de K^n no es única, de hecho en términos generales existen infinitas, si el espacio vectorial es infinito (hablamos de infinito en el sentido de cantidad de elementos no en el sentido de la dimensión), en nuestro caso por ser K^n un conjunto finito tiene un número finito de bases, más adelante se presentará un teorema al respecto.
2. La condición (1) nos dice que B genera a todo el espacio y esta forma de generar cada palabra es única (esto es algo que viene del álgebra lineal).
3. Todas las bases tiene el mismo número de palabras.
4. Esto se aplica también para cada subespacio vectorial, logrando así poder encontrar bases para nuestro fines a los códigos lineales.
5. Si S es un subconjunto tal que $S = \{\mathbf{0}\}$, entonces diremos que la base es vacío.

Definición 2.2.15. *Llamaremos **dimensión** del espacio vectorial al número de elementos que contiene la base B.*

Las definiciones 2.2.14 y 2.2.15 se pueden aplicar para subespacios que en nuestro caso serían códigos lineales, así una base para el código lineal C, es un subconjunto B' de C tal que las palabras de B' generan a cada palabra de C ($C =< B' >$), además las palabras de B' son linealmente independientes y la dimensión de C es la cantidad de palabras en B' ($\dim(C) = |B'|$). Con esto en mente veamos el siguiente resultado:

Teorema 2.2.16. *Un código lineal C de dimensión k contiene 2^k palabras de código.*

Demostración. Sea B la base de C, además como la dimensión de C es k, esto quiere decir que la cantidad de palabras que tiene B es k, así sea $B = \{v_1, v_2, \ldots, v_k\}$, además por definición de base tenemos:

$$\forall u \in C \text{ tal que } u = a_1v_1 + a_2v_2 + \cdots + a_kv_k, \text{ donde } a_i \in K \text{ para todo } i = 1, 2, \ldots, k.$$

Además por cada a_i tenemos dos opciones (cero y uno) y por el principio de la multiplicación tenemos 2^k combinaciones lineales diferentes, y por lo tanto tenemos 2^k palabras diferentes en C. □

Veamos ahora un resultado que relaciona las dimensiones del código lineal C y su dual. Este teorema es muy importante y suele aparecer en muchas áreas de álgebra en diferentes formas.

Teorema 2.2.17. *Sea $C =< S >$ un código lineal generado por S de K^n. Entonces:*

$$dim(C) + dim(C^{\perp}) = n.$$

Demostración. Sea $k = \dim(C)$, ahora analicemos por casos.

1. Si $k = 0$, en este caso tenemos que $C = \{\mathbf{0}\}$ y por lo tanto el dual es todo el espacio vectorial $C^{\perp} = K^n$, y es claro que $\dim(C^{\perp}) = n$ y en este caso se cumple la igualdad.

2. Si $k = n$ entonces $C = K^n$ y por lo tanto el dual es $C^{\perp} = \{\mathbf{0}\} \Rightarrow \dim(C^{\perp}) = 0$. Por lo tanto también se cumple la igualdad.

3. Si $0 < k < n$, en este caso formemos una matriz G de tal manera que las filas de G sean una palabra del conjunto S. La matriz G tiene n columnas ya que las palabras son de tamaño n.

   Ya que la dimensión de C es k entonces la matriz G tiene rango k, y por lo tanto cuando resolvamos el sistema $Gx^T = \mathbf{0}$ (se pone x^T ya que las palabras se toman en forma de fila), tendremos $n - k$ variables libres. Así sea la solución x_i tal que en la variable libre i (al ordenar las variables) toma el valor de 1 y en las demás variables libres 0, y como tenemos $n - k$ variables libres tenemos el siguiente conjunto solución con $n - k$ palabras $\{x_1, x_2, \ldots, x_{n-k}\}$. Es claro que el conjunto anterior es linealmente independiente por construcción (ya que en la palabra x_i tiene ceros en las posiciones donde aparecen las demás variables). Además cumplen que cualquier solución de $Gx^T = \mathbf{0}$ se escribe como una combinación lineal del conjunto anterior ya que al variar los valores de las variables libres tenemos las distintas soluciones, esto significa que:

$$x = a_1x_1 + a_2x_2 + \cdots + a_{n-k}x_{n-k}.$$

Por lo tanto tenemos que el conjunto $\{x_1, x_2, \ldots, x_{n-k}\}$ es un conjunto con palabras linealmente independientes y además es generador de las palabras que anulan a las palabras de C, por lo tanto por es una base de $C^\perp \Rightarrow \dim(C^\perp) = n - k$ y sustituyendo $k = \dim(C)$ y despejando n tenemos $\dim(C) + \dim(C^\perp) = n$.

Como vemos en todos los casos la suma de las dimensiones se cumple por lo tanto el teorema se ha demostrado. □

A continuación presentamos un teorema que nos permite saber cuantas bases tiene un espacio vectorial (para nuestro caso un código lineal), este resultado es sumamente importante ya que hace que se diferencie radicalmente entre trabajar con $\mathbb{R}^n$ que con K^n, ya que en el caso de del primero por ser un conjunto con infinitos elementos tiene infinitas bases cosa que no se tiene en K^n que por ser un conjunto finito tiene finitas bases.

Teorema 2.2.18. *Un código lineal C de dimensión k tiene precisamente* $\frac{1}{k!}\prod_{i=0}^{k-1}(2^k - 2^i)$ *diferentes bases.*

Demostración. Como cada palabra de C es candidato a estar en la base salvo la palabra cero, así primero tenemos $2^k - 1$ palabras para elegir la primera palabra en la base, la expresión anterior la podemos reescribir como $2^k - 2^0$.

Ahora luego de elegir la primera palabra, debemos elegir la segunda palabra base tendremos $2^k - 2^1$, esto es así porque debemos descartar todas las palabras que son generadas por la primera palabra base tomada y por el teorema 2.2.16. hay 2^1 palabras generadas.

Para la tercer palabra base tenemos $2^k - 2^2$ palabras para elegir ya que debemos descargar todas las palabras que pueden generar las dos primeras elegidas que por el teorema 2.2.16. existen 2^2 palabras generadas por ellas.

Así sucesivamente hasta completar las k palabras bases ya que el código lineal es de dimensión k, para ésta ultima elecciones tendremos $2^k - 2^{k-1}$, ya que debemos descartar las combinaciones lineales de las $k - 1$ palabras elegidas anteriormente.

Luego aplicamos el principio de la multiplicación y dividimos por $k!$ ya que no importa el orden en que hagamos la elección.

Con todo lo anterior tenemos:

$$\underbrace{\underline{2^k - 2^0} \times \underline{2^k - 2^1} \times \underline{2^k - 2^2} \times \ldots \times \underline{2^k - 2^{k-1}}}_{k \text{ espacios}} = \prod_{i=0}^{k-1}(2^k - 2^i)$$

Por lo tanto al dividir por $k!$ tendremos la cantidad de bases, que es $\frac{1}{k!}\prod_{i=0}^{k-1}(2^k - 2^i)$. □

Por último se hace el comentario que por la observación hecha en el ejemplo 2.2.11. existen códigos que verifican $C \subseteq C^{\perp}$. Así no se cumple en K^n que se pueda expresar como suma directa del código C y su dual, pero lo que sí sigue cumpliendo es: $dim(C) + dim(C^{\perp}) = n$, en K^n. Esto hace sumamente interesante nuestro espacio vectorial K^n.

2.3. ¿Cómo encontrar bases para C y su dual?

Para poder encontrar las bases para nuestros códigos lineales y sus duales respectivamente basta echar mano del álgebra lineal. Para el caso del código lineal $C =< S >$ es sumamente fácil, pues usamos la idea del álgebra lineal, la cual es:

Algoritmo 2.3.1. *Para encontrar una base para C, solo debemos formar una matriz A que contiene las palabras de código de S donde cada fila de A es una palabra de código de S. Luego a la matriz A haciendo operaciones por reglón la llevamos a la forma escalonada reducida. Entonces las filas distintas de cero de la matriz resultante formarán la base para C.*

Nota: El algoritmo anterior se puede aplicar para columnas en el sentido de hacer que cada columna de la matriz A tiene una palabra de código y luego es llevada a su forma escalonada reducida en las columnas, y las columnas distintas de cero son la base para C.

El algoritmo 2.3.1. funciona ya que lo único que hacemos es ir colocando múltiplos escalares de las filas que forman al código C y por lo tanto son palabras que seguirán generando al código lineal C.

Veamos un ejemplo donde apliquemos el algoritmo 2.3.1.

Ejemplo 2.3.2. *Usando el código lineal $C =< S >$ donde S es como en el ejemplo 2.2.11.*

Solución
Como $S = \{0101, 1010, 1100\}$, formemos la matriz A como en el algoritmo 2.3.1.

$$A = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \end{bmatrix} \longrightarrow \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 \end{bmatrix} \longrightarrow \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix} \longrightarrow \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix} \longrightarrow \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix}$$

Así la base de C es $B = \{1001, 0101, 0011\}$.

Ahora presentamos una forma para calcular la base del dual de $C =< S >$.

Algoritmo 2.3.3. *Para encontrar una base para $C^{\perp}$, aplicamos el algoritmo 2.3.1 a C de la matriz obtenida formamos la matriz G de $k \times n$ que contiene las filas distintas de cero de la matriz obtenida en el algoritmo 2.3.1. Luego formamos la matriz X de $k \times (n-k)$ de tal manera que contiene las columnas de G pero distintas de las columnas principales. Finalmente formamos una matriz H de $n \times (n-k)$ tal que:*

1. *En las filas de H se colocan las filas de X en el orden en que estaban las columnas principales en G respectivamente.*
2. *Por último en las $n-k$ filas de H restantes colocamos las filas de la matriz identidad I de $(n-k) \times (n-k)$ de forma ordenada en los espacios restantes.*

Las columnas de la matriz H forman la base para $C^{\perp}$.

El algoritmo 2.3.3. funciona ya que por el teorema 2.2.17 la dimensión del dual debe ser $n-k$ y como H por construcción tiene $n-k$ columnas (la cantidad de elementos de la base encaja), además $GH = \mathbf{0}$. Lo anterior es cierto porque si nosotros tomamos una permutación π tal que mueva las columnas de G y por lo tanto también las filas de H (esto es por como se construye H) y haga que las columnas principales de G estén al inicio y las filas principales de H estén abajo, tendremos:

$$\pi(GH) = [I, X] * [X, I]^T = I * X + X * I = X + X = \mathbf{0} \Rightarrow GH = \mathbf{0}.$$

Por lo tanto las columnas H anulan a la base de C y por ser $n-k$ palabras linealmente independiente en el dual de dimensión $n-k$ (por construcción ya que introducimos la matriz identidad I en H), tenemos entonces que las columnas de H forman una base para $C^{\perp}$.

Ahora aplicamos éste algoritmo a un ejemplo.

Ejemplo 2.3.4. *Encontrar la base del dual de código del ejemplo 2.3.2.*

Solución

De lo desarrollado en el ejercicio 2.3.2, tenemos que la matriz resultante del algoritmo 2.3.1 es:

$$\begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix}$$

De ésta matriz se verifica que la dimensión del código C es $k = 3$ y como las palabras tienen tamaño 4 esto quiere decir que $n = 4$ y por lo tanto la dimensión del dual de C es de $n - k = 4 - 3 = 1$.

Como vemos la matriz no tiene filas iguales a cero así la matriz G será la misma matriz:

$$G = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix}$$

Al eliminar las columnas principales tenemos que la matriz X es:

$$X = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

Como las columnas principales están en la columnas 1, 2 y 3, las filas de X se colocan de la manera en que la fila uno de X va en la fila 1 de H porque la primera columna principal está en la posición 1 y luego la segunda fila en la segunda (porque la segunda columna principal está en la posición 2) y así para la tercera es colocada en la tercera fila de H, por lo tanto tendremos que la matriz H es:

$$\begin{bmatrix} 1 \\ 1 \\ 1 \\ - \\ 1 \end{bmatrix} \longrightarrow H = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

Por lo tanto la base de $C^{\perp}$ es $B = \{1111\}$.

2.4. Matriz de codificación

Ya tenemos métodos sólidos para poder encontrar la base del código C y su dual, con esta nueva información podemos retomar nuestro objetivo de poder detectar y corregir errores en los códigos, pero para los códigos lineales.

Es necesario hacer énfasis que para nuestros códigos lineales hemos logrado obtener la siguiente información:

(i) Las palabras de C son de tamaño n (ya que $C \subseteq K^n$).

(ii) C tiene dimensión k (éste número representa la cantidad de palabras en su base como subespacio vectorial de K^n, además de decirnos que en la base hay k palabras base linealmente independientes).

(iii) C tiene una distancia d (éste número fue redefinido en la sección 2.1 como el menor peso de las palabras de código distintas de cero).

Así a un código C le llamaremos: código lineal (n, k, d). La terna proporciona información vital del código C.

La matriz que podemos formar con las palabras bases del código C tiene un papel muy importante en el trabajo de la codificación, así pues definámosla apropiadamente:

Definición 2.4.1. *Si C es un código (n, k, d), entonces alguna matriz cuyas filas formen una base para C es llamada **matriz generadora**.*

Observación: La matriz generadora del código C tiene k filas y n columnas, además el rango de la matriz es k.

Además si quisiéramos encontrar la matriz generadora de un código C, debemos considerar a nuestro código como el generado por sí mismo ($C =< C >$) y luego aplicamos el algoritmo 2.3.1 o si $C =< S >$ entonces solo aplicamos el algoritmo 2.3.1.

La razón del nombre es porque podemos generar cualquier palabra del código C mediante ésta matriz. Para ver esto veamos el siguiente teorema:

Teorema 2.4.2. *Si G es una matriz generadora de C, código lineal (n, k, d), entonces para cada palabra $v \in C$, existe una palabra u en K^k tal que $v = uG$, así C es el conjunto de todas las palabras $v = uH$ con $u \in K^k$. Además si $u_1G = u_2G$ si y solo si $u_1 = u_2$.*

Demostración. Como las filas de G forman una base para C, entonces sean $g_1, g_2, \ldots, g_k$ las filas de G entonces:

$$G = \begin{bmatrix} g_1 \\ g_2 \\ \vdots \\ g_k \end{bmatrix}$$

Ahora por ser base las palabras g_i tenemos que para todo $v \in G$ existen $a_1, a_2, \ldots, a_k$ dígitos tal que:

$$u = a_1g_1 + a_2g_2 + \cdots + a_kg_k = a_1a_2 \ldots a_k \begin{bmatrix} g_1 \\ g_2 \\ \vdots \\ g_k \end{bmatrix} = uG$$

Donde $u = a_1a_2 \ldots a_k$ y como vemos es de tamaño k por lo tanto $u \in K^k$. De hecho de lo anterior vemos que al recorrer sobre todas las palabras de K^k tendremos todas las combinaciones lineales de la base formada por los g_i, así C contiene todas las palabras de la forma $v = uG$, $u \in K^k$.

Finalmente recordemos que de la definición de base se tiene que la combinación lineal de las palabras bases es única para cada palabra de C así pues tenemos también que si $u_1G = u_2G$ si y solo si $u_1 = u_2$. □

El teorema 2.4.2 nos muestra que el código lineal (n, k, d) es el conjunto de todos los mensajes que se pueden mandar con las palabras de K^k pero codificadas. Esto quiere decir que el mensaje u lo podemos codificar como $v = uG$. También notemos que k dígitos son usados para información y el resto es redundancia, esto es cierto ya que al verificar la tasa de información tenemos:

$$\frac{\log_2(|C|)}{n} = \frac{\log_2(2^k)}{n} = \frac{k}{n}$$

La tasa de información nos dice que de los n dígitos de la palabra enviada k dígitos son de información y el resto es redundancia.

Para finalizar ésta parte veamos el siguiente ejemplo.

Ejemplo 2.4.3. *Sea C el código lineal $(5, 3, d)$ con matriz generadora:*

$$G = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \end{bmatrix}$$

Codificar la palabra $u = 010$.

Solución
Usando el teorema 2.4.2 tendremos:

$$uG = 010 \begin{bmatrix} 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \end{bmatrix} = 01010$$

Por lo tanto tenemos que la palabra $u = 010$ es codificada como 01010 en C.

2.5. Matriz de control de paridad

La matriz de control de paridad, está íntimamente ligada a la matriz generadora de un código, ya que la matriz de control de paridad corregirá las palabras que son afectadas por el ruido.

Definición 2.5.1. *Una matriz H es llamada **matriz de control de paridad** para un código lineal C, si las columnas de H forman una base para el código dual $C^\perp$.*

Observación: Para el código lineal (n, k, d) la matriz de control de paridad H, al tener en sus columnas una base para el código dual de C cumple que la dimensión que tiene es $n \times (n - k)$. Además para encontrar H basta aplicar el algoritmo 2.3.3 al código C.

Ahora veamos algunos resultados que son consecuencia de la definición 2.5.1

Teorema 2.5.2. *Si H es la matriz de control de paridad de C un código lineal (n, k, d), entonces C consiste de todas las palabras de $v \in K^n$ tal que $vH = \mathbf{0}$.*

Demostración. Supongamos que existe $v \in K^n$ que no pertenece a C pero cumple $vH = \mathbf{0}$, así sean $h_1, h_2, \ldots, h_{n-k}$ las columnas de H y por definición de matriz de control de paridad forman un base para $C^{\perp}$.

De lo anterior tenemos:

$$vH = v\begin{bmatrix} h_1 & h_2 & \ldots & h_{n-k} \end{bmatrix} = (v \cdot h_1)(v \cdot h_2)\ldots(v \cdot h_1) = 00\ldots0$$

Esto quiere decir que v es ortogonal a la base de $C^{\perp}$, pero si es ortogonal a la base del código dual entonces también lo será para todo el dual, así $v \in (C^{\perp})^{\perp}$, pero el dual del dual de un conjunto es el conjunto, por lo tanto $v \in C$ pero esto es una contradicción a lo supuesto al inicio, así C es el conjunto formado por todas las palabras $v \in K^n$ tal que $vH = \mathbf{0}$. □

Teorema 2.5.3. *H es la matriz de control de paridad de C si y solo si H^T es la matriz generadora de $C^{\perp}$.*

Demostración. Si H es la matriz de control de paridad de C por definición sus columnas forma una base para el dual de C, así al tomar la transpuesta tendremos que H^T tiene en sus filas una base para el dual, pero esto es la definición de matriz generada por lo tanto H^T es la matriz generadora de $C^{\perp}$.

Ahora si H^T es la matriz generadora de $C^{\perp}$ sus filas son base para el dual de C, entonces al tomar H tendrá que sus columnas forman la base para $C^{\perp}$, pero esto es la definición de matriz de control de paridad, por lo tanto H es la matriz de control de paridad de C. □

Veamos como encontrar una matriz de control de paridad para un caso en particular.

Ejemplo 2.5.4. *Encontrar la matriz de control de paridad de C del ejemplo 2.4.3.*

Solución
Del ejemplo 2.4.3 la matriz generadora de C es:

$$G = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \end{bmatrix}$$

Ahora aplicando el algoritmo 2.3.3 a ésta matriz y tendremos que la matriz H es:

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

Notemos que si tomamos la palabra base 10011 da la palabra cero en K^2.

$$H = 10011 \begin{bmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} = 00$$

Y esto se verifica para toda palabra de C.

2.6. Códigos sistemáticos y códigos equivalentes

Los códigos sistemáticos tienen ventaja sobre el resto de códigos lineales, la razón está ventaja será desarrollada a continuación.

Definición 2.6.1. *Una matriz A de $n \times m$ con $n \leq m$ diremos que está en **forma estándar**, si $A = [I_n, X_{n\times(m-n)}]$, donde I_n es la identidad de $n \times n$ y $X_{n\times(m-n)}$ es una matriz de $n \times (m-n)$.*

Definición 2.6.2. *Un código lineal C se llamará **código sistemático**, si la matriz generadora de C está en forma **estándar***

Observación: No todos los códigos son sistemáticos, por ejemplo si consideramos el código generado por la matriz:

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

Como vemos la matriz G no tiene la identidad I_2 en sus primeras dos columnas, por lo tanto el código C generado por G no es un código sistemático.

Ejemplo 2.6.3. *El código del ejemplos 2.4.3 es generado por la matriz:*

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

Como vemos la matriz G tiene la identidad I_3 en sus primeras tres columnas por lo tanto el código generado por G en éste caso sí es sistemático.

La ventaja de tener un código sistemático radica en que es fácil saber cuales son los dígitos de información en una palabra; de hecho hasta éste momento nunca se sabía cuales eran realmente los dígitos de información y cuales los de redundancia. El siguiente teorema formaliza éste hecho.

Teorema 2.6.4. *Si C es un código lineal (n, k, d) con matriz generadora G en forma estándar, entonces los primeros k dígitos en la palabra de código $v = uG$ son de la palabra u de K^k.*

Demostración. Como la matriz G está en su forma estándar entonces $G = [I_k, X]$, donde X es una matriz de tamaño $k \times (n-k)$.

Además como $v = uG$ donde $u \in K^k$, esto por el teorema 2.4.2, así tenemos:

$$v = uG = u[I_k, X] = [uI_k, uX] = [u, uX]$$

De lo anterior es fácil ver que $u \in K^k$, entonces u tiene tamaño k, además que u está en las primeras k columnas de uG, así los primeros k dígitos de v son de u. □

Observación: El teorema 2.4.2 nos dice que si queremos mandar una mensaje u, debemos primero codificarlo de la forma $v = uG$, donde G es la matriz generadora del código C en forma estándar, entonces si IMLD concluye que v fue enviado, entonces el mensaje que queríamos es u; en otras palabras logrando tener v la palabra que fue enviada es sumamente fácil saber cuales son los dígitos de información en el caso de tener una matriz en forma entandar serán siempre los primeros k dígitos.

Una última observación es que para un código lineal (n, k, d), los últimos $n-k$ dígitos son de redundancia.

Ejemplo 2.6.5. *Para el código del ejemplo 2.6.3, nosotros deseamos mandar el mensaje $u = 101$ a la hora de codificarlo tendremos:*

$$101G = 101 \begin{bmatrix} 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \end{bmatrix} = 10110$$

Por lo tanto $10110 = [u10]$, donde el mensaje es 101.

Es muy útil tener un código sistemático pero qué sucede si tenemos un código que no es sistemático, cómo lograr conseguir un código sistemático a partir del que ya tenemos. Para ello veamos que significa tener dos códigos equivalentes.

Definición 2.6.6. *Dos códigos C y C' son equivalentes, si las palabras de C' se obtienen de C permutando las posiciones de los dígitos de las palabras de C, ésta permutación debe aplicarse a todas las palabras de C.*

El siguiente teorema nos permite encontrar códigos sistemáticos a partir de otros códigos.

Teorema 2.6.7. *Un código lineal C es equivalente a algún código lineal sistemático C'.*

Demostración. Sea C un código lineal, usando el algoritmo 2.3.1, podemos encontrar una matriz generadora G en forma escalonada reducida. Ahora si G ya tiene la forma estándar entonces $C' = C$, pero si G no está en la forma estándar entonces creamos la matriz G' tal que se permutan las columnas principales de tal manera que G' esté en forma estándar (al permutar los dígitos siguen siendo linealmente independiente las filas), así C es equivalente al código generador por la matriz G'. □

Así con éste último teorema, la demostración nos proporciona una forma de encontrar códigos lineales sistemáticos a partir de uno que no lo sea.

Para el caso del código que se genera con la matriz:

$$G = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Utilizando el teorema 2.6.7 y los pasos de su demostración logramos concluir que C generado por G es equivalente al código C' generado por G', donde:

$$G' = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

Donde solo se permutan las columnas 2 y 3.

Como vemos es sumamente más práctico trabajar con códigos sistemáticos, tanto para encontrar los dígitos de información como para encontrar la matriz de control de paridad ya que si $G = [I, X]$, entonces por el algoritmo 2.3.3 y tendremos que H es:

$$H = \begin{bmatrix} X \\ I \end{bmatrix}.$$

Los cálculos se reducen para códigos lineales, siempre que sea posible trabajaremos con los códigos lineales sistemáticos. Además las propiedades más importantes se preservan entre códigos equivalentes como veremos a continuación:

Teorema 2.6.8. *Los códigos equivalentes tienen la misma dimensión y distancia.*

Demostración. Sea un código lineal C con dimensión k y distancia d, además sea G la matriz generadora de C, por definición la dimensión de G es $k \times n$. Sea C' un código equivalente a C, una matriz generadora para C' es G' tal que se permutan los dígitos de las filas de G, como esta acción no altera que las filas sean linealmente independiente tendremos que G' es de tamaño $k \times n$, por lo tanto C' tiene también dimensión k. Finalmente sea v la palabra de código de C tal que $wt(v) = d$, como la diferencia entre las palabras de C con las de C' es que las palabras de C' tienen los dígitos permutados, la acción de permutar dígitos no cambia el peso de las palabras así la palabra generada a partir de v para C' tiene peso igual a d; $wt(v') = d$, por lo tanto la distancia del código C' es también d. □

De lo demostrado anteriormente hace que al trabajar con códigos sistemáticos se equivalente a trabajar con cualquier otro código equivalente al sistemático.

2.7. Clases laterales

En ésta sección comenzaremos a construir las herramientas que nos permitirá más adelante poder corregir las palabras recibidas.

Definamos así la clase lateral de una palabra u.

Definición 2.7.1. *Sea C un código lineal de K^n y sea $u \in K^n$, definimos la **clase lateral de C determinado por u** como el conjunto de todas las palabras de la forma $v + u$, para cada $v \in C$. Esta clase lateral se denota por $C + u$, así:*

$$C + u = \{v + u : v \in C\}$$

Ejemplo 2.7.2. *Encontrar la clase lateral de $u = 101$, para el código $C = \{000, 111\}$.*

Solución
Para encontrar $C + 101$, solo aplicamos la definición 2.7.1, así tenemos:

$$C + 101 = \{000 + 101, 111 + 101\} = \{101, 010\}.$$

Por lo tanto $C + 101 = \{101, 010\}$.

Además notemos que $C + 111 = C$:

$$C + 111 = \{000 + 111, 111 + 111\} = \{111, 000\} = C.$$

También que $C + 010 = C + 101$:

$$C + 010 = \{000 + 010, 111 + 010\} = \{010, 101\} = C + 101$$

Como vemos una de las clases laterales es el mismo código C, además la clase lateral $C + 010$ no es afectado por quien lo determina, así una misma clase puede ser determinada por muchos elementos. Para formalizar estos resultados particulares presentamos el siguiente teorema.

Teorema 2.7.3. *Sea C un código lineal (n, k, d) y sean u y v palabras de tamaño n. Entonces las clases laterales cumplen:*

1. *Si $u \in C + v$, entonces $C + u = C + v$; es decir, cada palabra en la clase lateral determina esa clase lateral.*
2. *La palabra u está en la clase lateral $C + u$.*
3. *Si $u + v \in C$, entonces u y v están en la misma clase lateral.*
4. *Si $u + v$ no está en C, entonces u y v están en diferentes clases laterales.*

5. *Cada palabra en K^n está contenida en una y solo una clase lateral de C; es decir, ya sea $C + u = C + v$, o $C + u$ y $C + v$ no tienen palabras en común.*
6. *$|C + u| = |C|$; es decir, el número de palabras en una clase lateral de C es igual al número de palabras en el código C.*
7. *Si C tiene la dimensión k, entonces existen exactamente 2^{n-k} diferentes clases laterales, y cada clase lateral contiene exactamente 2^k palabras.*
8. *El código C es también una clase lateral.*

Demostración. Demostramos cada numeral en orden:

1. Sea $u \in C + v$, entonces $u = w + v$ para algún $w \in C$, así:

$$\begin{aligned} C + u &= \{h + u : h \in C\} \\ &= \{h + w + v : h \in C\} \\ &= \{(h + r) + v : h \in C\} \\ &= \{r + v : r \in C\} \\ &= C + v \end{aligned}$$

   Por lo tanto $C + u = C + v$.

2. Como C es lineal entonces $\mathbf{0} \in C$, así $u = \mathbf{0} + u \in C + u$, por lo tanto $u \in C + u$.
3. Ya que $u + v \in C$, entonces existe $w \in C$ tal que $u + v = w \Rightarrow u = w + v \in C + v \Rightarrow u \in C + v$, pero por (1), tenemos que $C + u = C + v$; lo que quiere decir que u y v están en la misma clase lateral.
4. Supongamos por contradicción que u y v están en la misma clase lateral, así $u, v \in C + w$, entonces existen $h, r \in C$, tal que $u = h + w$ y $v = r + w$, así al tomar la suma de u y v, tendremos:

$$u + v = (h + w) + (r + w) = h + r + (w + w) = h + r \in C \Rightarrow u + v \in C.$$

   Pero lo anterior es una contradicción ya que por hipótesis tenemos que la suma no está en C. Por lo tanto u y v están en distintas clases laterales.

5. Dos clases laterales determinada por dos palabras distintas $u \neq v$, cumplen:

$$(C + u) \cap (C + v) = \varnothing \text{ o } (C + u) \cap (C + v) \neq \varnothing.$$

   En el caso $(C + u) \cap (C + v) = \varnothing$, entonces las clases laterales $C + u$ y $c + v$ no tienen palabras en común.

   Ahora si $(C + u) \cap (C + v) \neq \varnothing \Rightarrow$ existe w tal que $w \in C + u$ y $w \in C + v$ y aplicando el resultado (1) tendremos:

$$C + u = C + w = C + v \Rightarrow C + u = C + v.$$

6. La igualdad $|C + u| = |C|$ es cierta porque la clase lateral $C + u$ tiene tantos elementos como elementos tiene C ya que en $C + u$ los elementos son de la forma $v + u$ y u se mantiene fijo u y variando v en C; lo que significa que por cada v en C se hace una suma en $C + u$.

7. Usando (5) y (6) tenemos que cada clase lateral tiene 2^k y además las palabras de K^n están en una y solo una clase lateral, así basta hacer una división para saber cuantas clases laterales podremos tener, así la cantidad de clases laterales en K^n es:

$$\frac{2^n}{2^k} = 2^{n-k}$$

8. C es una clase lateral ya que si tomamos la clase lateral determinado por $\mathbf{0}$ tendremos:

$$\begin{aligned} C + \mathbf{0} &= \{v + \mathbf{0} : v \in C\} \\ &= \{v : v \in C\} \\ &= C. \end{aligned}$$

Por lo tanto C es una clase lateral.

□

Finalmente veamos un ejemplo:

Ejemplo 2.7.4. *Para el código* $C = \{0000, 1011, 0101, 1110\}$*, encontrar sus clases laterales en* K^4.

Solución
Aplicando la definición 2.7.1 y el teorema 2.7.3, tendremos que las clases laterales son:

$$\begin{aligned} C &= \{0000, 1011, 0101, 1110\} \\ C + 1000 &= \{1000, 0011, 1101, 0110\} \\ C + 0100 &= \{0100, 1111, 0001, 1010\} \\ C + 0010 &= \{0010, 1001, 0111, 1100\} \end{aligned}$$

En la siguiente sección podremos aplicar las clases laterales para poder corregir las palabras que son afectadas por el ruido.

2.8. MLD para códigos lineales

Como logramos observar en el capítulo 1 al estar realizando tablas de MLD es sumamente tedioso y poco práctico, el objetivo que buscamos es poder encontrar una manera eficiente para ello y de hecho los códigos lineales permiten hacer esta labor más fácil haciendo uso de la matriz de control de paridad y las clases laterales podremos mostrar un mecanismo óptimo para lograr superar la barrera de tener que revisar palabra por palabra.

Recordemos que si mandamos la palabra $v \in C$ y se recibe w, entonces tenemos el patrón de error $u = v + w$, despejando v tendremos $u + w = v$, pero $v \in C$, entonces $u + w \in C$ y por (3) del teorema 2.7.3 tendremos que u y w están en la misma clase lateral; de hecho todo patrón de error para w estarán en su clase lateral y no en otra.

Así solo basta tomar la clase lateral $C + w$ para encontrar el patrón de error, luego debemos recordar que los patrones de error con peso más pequeño son los patrones de error más propensos a suceder así elegimos u de $C + w$ con el peso menor, entonces podemos concluir que la palabra que fue enviada es $v = u + w$.

Veamos un ejemplo donde podamos aplicar la idea anterior.

Ejemplo 2.8.1. *Sea* $C = \{0000, 1011, 0101, 1110\}$, *encontrar cual fue la palabra* v *de* C *más probable de haber sido enviado si se recibe* $w = 1101$.

Solución
Primero calculamos las clases laterales de C. Las clases laterales las colocamos en columnas de la siguiente manera:

0000	1000	0100	0010
1011	0011	1111	1001
0101	1101	0001	0111
1110	0110	1010	1100

La clase lateral donde está contenida la palabra $w = 1101$ es la segunda columna, y de ahí tomamos la palabra de menor peso la cual es 1000, así por IMLD o CMLD tendremos que v es:

$$v = u + w = 1000 + 1101 = 0101$$

Así la palabra más probable de haber sido envíada es 0101.

Con ésta idea aún no es fácil la detección del error. Para poder usar la matriz de control de paridad primero definiremos el síndrome de una palabra.

Definición 2.8.2. *Sea C un código lineal (n, k, d) y H la matriz de control de paridad de C, entonces para la palabra $w \in K^n$ llamaremos* ***síndrome*** *de w a la palabra wH en K^{n-k}.*

Ejemplo 2.8.3. *Usando el código C del ejemplo 2.8.1 para encontrar el síndrome de $w = 1101$.*

Solución
La matriz de control de paridad de C es:

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

Ahora aplicando la definición 2.8.2, con esto tendremos:

$$wH = 1101 \begin{bmatrix} 1 & 1 \\ 0 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} = 11$$

Por lo tanto el síndrome de $w = 1101$ es 11. De hecho si calculamos el síndrome al patrón de error de w tendremos:

$$uH = 1000 \begin{bmatrix} 1 & 1 \\ 0 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} = 11$$

Así el síndrome del patrón de error es 11. Observemos que en el ejemplo se concluye que la palabra más probable de haber sido enviada se encuentra con w y u, de la manera siguiente $v = 1101 + 1000 = 0101$, lo que quiere decir que por el patrón de error, el error ocurre en el primer dígito; y de hecho el síndrome ocupa la posición 1 en H.

Esto quiere decir que hay una relación entre el síndrome y los dígitos que son afectados por el ruido. Con esto en mente tenemos el siguiente teorema.

Teorema 2.8.4. *Sea C un código lineal (n, k, d) y H la matriz de control de paridad de C. Sean w y u palabras en K^n. Entonces se verifica:*

1. *$wH = \mathbf{0}$ si y solo si w es una palabra de código en C.*
2. *$wH = uH$ si y solo si w y u se encuentran en la misma clase lateral de C.*
3. *Si u es el patrón de error para una palabra recibida w, entonces uH es la suma de las filas de H que corresponden a las posiciones en las que se produjeron errores en la transmisión.*

Demostración. Demostremos en orden cada numeral.

1. Para la primera implicación tenemos que por el teorema 2.5.1 C contiene todas las palabras de $v \in K^n$ tal que $vH = \mathbf{0}$, así como w cumple $wH = \mathbf{0}$, entonces por el teorema 2.5.1 $w \in C$.

   Para la otra implicación es trivial pues si $w \in C$ y como H es l matriz que tiene la base del código dual de C tenemos que $wH = \mathbf{0}$.

2. Para la primera implicación tenemos que si

$$wH = uH \Rightarrow wH + uH = \mathbf{0} \Rightarrow (w + u)H = \mathbf{0}$$

   Y por (1) tenemos que $w + u \in C$, y por (3) del teorema 2.7.3 w y u están en la misma clase lateral de C

   Ahora para la segunda implicación si w y u están en la misma clase lateral de C, tenemos $w \in C + u$, entonces existe $h \in C$ tal que $w = h + u$, ahora despejando h tendremos:

$$w + u = h \in C \Rightarrow w + u \in C$$

   Y por (1) tenemos:

$$(w + u)H = \mathbf{0} \Rightarrow wH + uh = \mathbf{0} \Rightarrow wH = uH$$

3. Ya que u es el patrón de error de la palabra recibida w, tenemos que cada uno en sus dígitos representa que en esa misma posición en w hay un dígito incorrecto, por lo tanto cuando hacemos el producto uH hacemos que las filas que corresponden a la misma posición de los unos se sumen, así el síndrome será una palabra que es la suma de las filas de H donde ocurre el error.

□

Observación: Si en la transmisión no ocurre error entonces la palabra recibida w cumple $wH = \mathbf{0}$, pero que w cumpla $wH = \mathbf{0}$ no implica que no hayan ocurrido errores en la transmisión.

Además las palabras en la misma clase lateral tiene el mismo síndrome. Esto quiere decir que si queremos calcular el síndrome de una clase lateral basta tomar cualquier palabra de dicha clase y hacemos wH y éste será el síndrome de las palabras dentro de la clase lateral de w.

También es necesario mencionar que por todo lo anterior discutido elegimos como representante de la clase lateral la palabra de menor peso (en caso de haber más de uno con el mismo peso se elegirá de manera arbitrario el represéntate si estamos usando CMLD, en

caso contrario si usamos IMLD ésta clase no se toma en cuenta y en este caso la palabra w se pide retransmisión); éste será de hecho el patrón de error para todas las palabras en la clase lateral, y usamos éste también para calcular el síndrome de la clase lateral (esto nos dirá que posiciones los dígitos están errados y por ende deben ser cambiados).

Así veamos un último ejemplo donde aplicamos todo lo anterior discutido.

Ejemplo 2.8.5. *Sea C el código del ejemplo 2.8.1. Calcular los síndromes para el caso de CMLD y luego verifique cuál es la palabra más probable de haber sido enviado si se recibe* $w = 1011$

Solución
De lo desarrollado en los ejemplos 2.8.1 y 2.8.3 tenemos que los representantes de las clases y la matriz de control de paridad son:

Los representantes: 0000, 1000, 0100 y 0010.

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

Ahora formamos una tabla donde tengamos los representantes y los síndromes.

u representantes de las clases laterales	uH síndrome
0000	00
1000	11
0100	01
0010	10

La tabla anterior es llamada **matriz de decodificación estándar** o **SDA** por sus siglas en inglés **standard decoding array**. Ya con ésta información veamos el síndrome de 1011, para encontrar v, así:

$$wH = 1011 \begin{bmatrix} 1 & 1 \\ 0 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} = 00$$

La tabla Como el síndrome es 00 entonces el patrón de error u es 0000, así la palabra más probable de haber sido enviada es $v = 1011 + 0000 = 1011$.

Aunque luego de pasar por todo el proceso de obtener los síndromes y la matriz de control de paridad las operaciones son sumamente fáciles, como veremos más adelante aún sigue siendo tedioso los cálculos, aunque desde otro punto de vista, sin embargo hemos

logrado mucho ya que lo que los códigos lineales permiten es crear estos procesos que son más fáciles de programar y aplicar de manera computacional, así como crear códigos con ciertas especificaciones para poder detectar y corregir errores de manera más óptima para los procesos descrito a lo largo de este capítulo. Y al final de cuentas esto es algo importante para la teoría de código saber elegir un código C que nos facilite nuestro trabajo en los aspectos necesarios. En el próximo capítulo seguiremos estudiando la estructura de los códigos lineales mejorando los métodos ya descritos en éste capítulo.

2.9. Distancia y confiabilidad de un código lineal

Además de definir la distancia de un código como el menor peso de una palabra distinta de la palabra cero en el código lineal, también podemos calcular la distancia a partir de la matriz de control de paridad H, así definamos el siguiente teorema.

Teorema 2.9.1. *Sea H la matriz de control de paridad del código lineal C. Entonces C tiene distancia d si y sólo si todos los conjuntos de $d-1$ filas de H son linealmente independientes y al menos un conjunto de d filas de H son linealmente dependientes.*

Demostración. Primero denotemos la matriz H como:

$$H = \begin{bmatrix} h_1 \\ h_2 \\ \vdots \\ h_n \end{bmatrix}$$

Con esto analicemos por doble implicación:

$\Rightarrow$ Asumamos primero que la distancia del código lineal C es d, entonces existe un palabra de código de C con peso igual a d ($wt(v) = d$), esto quiere decir que v tiene d unos y además es la menor cantidad de unos que tienen las palabras de C. Así tendremos:

$$vH = \mathbf{0} \Rightarrow a_1h_1 + a_2h_2 + \cdots + a_nh_n = \mathbf{0}, \text{ donde } v = a_1a_2 \ldots a_n.$$

Pero como $wt(v) = d$ significa que d de los a_i son unos y por lo tanto por la definición 2.2.12 $h_1, h_2, \ldots, h_n$ son linealmente dependientes de hecho al tomar las filas donde se multiplica por uno tendremos un conjunto de d filas y estas serán linealmente dependientes; ya que seguirán sumando la palabra cero (porque las filas restantes se multiplican por cero). Por otro lado no puede existir un conjunto de $d-1$ filas que sean linealmente dependientes ya que al hacer un análisis parecido al anterior tendremos que existe una palabra de código u con peso $wt(u) = d-1$ tal que $uH = \mathbf{0}$ y por el teorema 2.5.2. $u \in C$ y eso será una contradicción ya que la distancia de C es d y no pueden haber palabras distintas a la palabra cero con peso menor a d en C. Por lo tanto todo conjunto de $d-1$ filas de H serán linealmente independientes.

$\Leftarrow$ Ahora como todo conjunto de $d-1$ filas de H son linealmente independientes, tenemos que no existe $v \in K^n$ con peso $wt(v) = d-1$ tal que $vH = \mathbf{0}$ y por lo tanto ninguna palabra de C tiene peso $d-1$ o menos que $d-1$ (por el teorema 2.5.2.), por otro lado como existe un conjunto de d filas que son linealmente dependientes esto quiere decir que existe una palabra v con peso $wt(v) = d$ tal que $vH = \mathbf{0}$ y por el teorema 2.5.2 $v \in C$ y por lo tanto al no poder existir una palabra (distinta a la palabra cero) con peso menor a d, la distancia de C es d.

Lo anterior completa la demostración. □

El resultado anterior nos permite conocer más a un código lineal dada su matriz de control de paridad, de hecho podemos saber los parámetros que definen un código lineal; los cuales son el tamaño de las palabras n, la dimensión del código lineal k y la distancia del código d. En otras palabras dada la matriz de control de paridad de un código lineal C, el código C queda perfectamente determinado como un código lineal (n, k, d).

Por último veamos que saber el grado de confiabilidad de un código lineal C es sumamente fácil de calcular.

Recordemos que en el capítulo 1 vimos que para calcular $\theta_p(C, v)$ es necesario calcular primero $L(v)$ el conjunto de las palabras que están más cerca de v que de otra palabra de código. También en las secciones anteriores vimos que si la palabra se encuentra más cerca de una y solo una palabra de código es porque están en una clase lateral con un único líder esto quiere decir que podemos redefinir $L(v)$, como:

$$L(v) = \{w : w = v + u, \text{ donde } u \text{ es el único líder de la clase de } w\}.$$

Otra cosa interesante es la probabilidad de que v se enviado cuando se recibe w es $\phi_p(w, v)$ pero este número depende estrictamente del patrón de error u, esto quiere decir que por ende al saber cuales son los líderes de las clases (excluimos las clases donde exista más de un líder) podemos calcular $\phi_p(w, v)$ sin importar la palabra v que se envió (porque solo debemos encontrar el líder de la clase de w y calcular la probabilidad), así $\theta_p(C, v)$ no depende de v y lo reescribimos $\theta_p(C)$, ahora como los líderes deben ser únicos y son los de menor peso en cada clase lateral tenemos que estas palabras están en el conjunto $L(\mathbf{0})$. Por lo tanto la confiabilidad de C se calcula.

$$\theta_p(C) = \sum_{u \in L(\mathbf{0})} p^{n-wt(u)}(1-p)^{wt(u)}. \tag{2.1}$$

La ecuación 2.1 es la confiabilidad que un código lineal C tiene o la probabilidad de que podamos saber cual fue la palabra enviada realmente, que en caso de ser un código lineal resulta fácil su cálculo pues solo debemos concentrar nuestra atención a los líderes de las clases laterales (siempre que sean únicos). De hecho el conjunto de patrones de errores que

puede corregir usando IMLD es el mismo número de sumando que tendremos en 2.1.

Las ventajas de trabajar con códigos lineales son muchas ya que tienen más estructura matemática facilitando algunos procesos. De aquí en adelante trabajaremos con códigos lineales dedicando nuestra atención a un tipo especifico de código.

Capítulo 3

Códigos perfectos

En éste último capítulo vamos a estudiar un hecho que hasta el momento no ha sido nuestro interés y es la pregunta ¿cuántas palabras debe tener el código C de tal manera que pueda corregir t errores?, responder ésta pregunta nos llevará definir ciertos tipos de límites tanto superiores como inferiores para la cantidad de palabras que deben ir en C. También al definirles estos límites resultarán ciertos tipos de códigos muy importantes entre los códigos correctores de errores. Se concentrará el estudio en los códigos perfectos, con el objetivo de crear algoritmos más eficientes que los expuestos en el capítulo 2 para la codificación y decodificación, pero sobre todo para poder corregir un máximo de tres errores.

3.1. Límites para los códigos

En esta sección estudiaremos la cantidad máxima de palabras que puede tener un código lineal en función del tamaño de las palabras n y la distancia del código d. Aunque el problema de saber cuántas palabras debe tener un código lineal, en general no está resuelto, se presentarán algunos resultados para ciertos valores de n y d, necesarios para lograr definir adecuadamente los códigos perfectos.

A continuación presentamos la primera definición.

Definición 3.1.1. $\binom{n}{t}$ *es el número de palabras de tamaño n y peso t.*

La definición anterior es coherente ya que si queremos saber cuántas palabras de peso t existen en K^n, debemos de calcular las maneras que se puede elegir t posiciones (sin importar el orden) de las n que tiene cada palabra en K^n (las posiciones es una referencia a los lugares donde van los dígitos), y esto se calcula usando el combinatorio $\binom{n}{t}$.

Usando la definición 3.1.1 podemos demostrar el siguiente teorema.

Teorema 3.1.2. *Si $0 \leq t \leq n$ y v es una palabra de tamaño n, entonces el número de palabras de tamaño n que están a una distancia a lo sumo t de la palabra v es precisamente:*

$$\binom{n}{0} + \binom{n}{1} + \binom{n}{2} + \cdots + \binom{n}{t}.$$

Demostración. Definamos primero el conjunto A_v para la palabra v de la siguiente manera:

$$A_v = \{w \in K^n : d(v, w) \leq t\}$$

Ahora usemos la definición de distancia para tener una forma más precisa del número de elementos en A_v.

$$\begin{aligned} A_v &= \{w \in K^n : d(v,w) \leq t\} \\ &= \{w \in K^n : wt(v+w) \leq t\} \\ &= \{u \in K^n : wt(u) \leq t\}; \text{ donde } u = v + w \\ &= \{u \in K^n : wt(u) = 0\} \cup \{u \in K^n : wt(u) = 1\} \cup \cdots \cup \{u \in K^n : wt(u) = t\} \end{aligned}$$

Cada conjunto $\{u \in K^n : wt(u) = i\}$ para $i = 0, 1, 2, \ldots, t$, es disjunto y además por la definición 3.1.1 tienen cardinalidad $\binom{n}{i}$. Así usando el principio de la suma tendremos:

$$|A_v| = \binom{n}{0} + \binom{n}{1} + \binom{n}{2} + \cdots + \binom{n}{t}$$

Lo anterior termina la demostración ya que A_v tiene las palabras que están a una distancia máxima t de v. □

De aquí en adelante cuando se diga: C es un código de tamaño n, se hará referencia a que las palabras que tiene C son de tamaño n ($C \subseteq K^n$). Con lo anterior estamos listos para demostrar uno de los tres teoremas más importantes de esta sección.

Teorema 3.1.3. ***(Límite de Hamming)*** *Si C es un código de tamaño n y distancia $d = 2t + 1$ o $d = 2t + 2$, entonces*

$$|C| \leq \frac{2^n}{\binom{n}{0} + \binom{n}{1} + \cdots + \binom{n}{t}}.$$

Demostración. Primero volvamos a definir el conjunto A_v como se hizo en la demostración del teorema 3.1.2 para v_1 y v_2 palabras de código distintas de C, y demostremos que:

$$A_{v_1} \cap A_{v_2} = \emptyset.$$

Para verificar esto procedamos por contradicción, así digamos que existe $w \in A_{v_1} \cap A_{v_2}$, esto significa que w está a una distancia a lo sumo t de ambas palabras de código que matemáticamente es:

$$d(w, v_1) \leq t \text{ y } d(w, v_2) \leq t.$$

Ahora usando la desigualdad triangular en $d(v_1, v_2)$, con esto:

$$d(v_1, v_2) \leq d(v_1, w) + d(w, v_2) \leq t + t = 2t \Rightarrow d(v_1, v_2) < 2t + 1 \text{ o } d(v_1, v_2) < 2t + 2.$$

Pero lo anterior es una contradicción ya que en ambos casos la distancia entre v_1 y v_2 es menor que la distancia del código y por definición d es la menor distancia entre dos palabras de código de C. Por lo tanto $A_{v_1} \cap A_{v_2} = \emptyset$.

Así los conjuntos A_v para todo $v \in C$ no tienen elementos en común, y con ello quiere decir que por cada conjunto A_v tenemos una palabra de C, además en K^n existen 2^n palabras. Finalmente por el teorema 3.1.2 tenemos que cada conjunto A_v tienen $\binom{n}{0} + \binom{n}{1} + \cdots + \binom{n}{t}$ palabras. Esto quiere decir que todos los A_v tienen la misma cantidad de elementos, además que cada palabra de código está en cada conjunto A_v, así al multiplicar $|A_v|$ por la cantidad de palabras del código lineal, tendremos a lo sumo 2^n palabras (ya que no podemos salirnos del espacio de trabajo K^n).

$$|C| \left[\binom{n}{0} + \binom{n}{1} + \cdots + \binom{n}{t} \right] \leq 2^n.$$

Así por lo tanto tendremos:

$$|C| \leq \frac{2^n}{\binom{n}{0} + \binom{n}{1} + \cdots + \binom{n}{t}}.$$

Esto termina la prueba. □

Del teorema demostrado anteriormente observemos lo siguiente:

1. El teorema 3.1.3 conocido por límite de Hamming es aplicable tanto a un código lineal o no.

2. Si el código C tiene una distancia $d = 2t + 1$, tendremos $t = (d - 1)/2$ y por el teorema 1.8.6 el código C corrige todos los patrones de error de peso menor o igual que t.

3. Cuando apliquemos el teorema 3.1.3 en el caso de querer un código lineal debemos buscar la potencia de dos más grande que verifique la desigualdad para la cardinalidad del código. En el caso que solo deseemos un código cualquiera tomaremos el mayor entero que cumpla la desigualdad para la cantidad de elementos del código.

Ahora veamos un ejemplo donde apliquemos el límite de Hamming.

Ejemplo 3.1.4. *Encontrar el límite de Hamming para el código C con los siguientes parámetros $n = 8$ y $d = 3$.*

Solución
Calculamos la fracción $\frac{2^n}{\binom{n}{0}+\binom{n}{1}+\cdots+\binom{n}{t}}$ usando los parámetros dados.

Como $d = 3$ quiere decir que $t = 1$, así tendremos:

$$\frac{2^8}{\binom{8}{0}+\binom{8}{1}} = \frac{2^8}{1+8} = \frac{256}{9} \approx 28{,}44.$$

Y por el teorema 3.1.3 tenemos que:

$$|C| \leq 28{,}44.$$

Así para el caso de tener un código no lineal C de tener a lo sumo 28 palabras ($|C| \leq 28$). Por otro lado si deseamos que C sea un código lineal la cantidad de elementos en C es una potencia de 2, así la potencia más grande que cumple ser menor que 28,44 es 2^4, así $|C| \leq 2^4$, y por lo tanto también en ese caso la dimensión de C, $k \leq 4$. Finalmente el límite de Hamming es 256/9.

Ahora veremos otro límite para la cantidad de elementos de un código y de hecho el siguiente teorema solo es aplicable a un código lineal, así presentamos el segundo teorema más importante de está sección.

Teorema 3.1.5. ***(Límite de Singleton)*** *Para algún código lineal (n, k, d), $d - 1 \leq n - k$.*

Demostración. Dado que el código lineal tiene parámetros (n, k, d), sabemos que la matriz de control de paridad es de dimensión $n \times (n - k)$. Además al tener el código distancia d, todos los conjuntos con $d - 1$ filas de la matriz de paridad son *l.i.* (linealmente independiente).

Por otro lado las filas de la matriz de control de paridad son de longitud $n - k$, entonces el espacio vectorial K^{n-k} tiene $n - k$ palabras *l.i.* en la base, y de hecho no pueden existir más palabras *l.i.*

Así uniendo los argumentos anteriores tenemos que $d - 1 \leq n - k$, ya que no puede suceder que $d - 1 > n - k$, porque esto haría que existan más palabras *l.i.* de las que hay en la base y eso es imposible en un espacio vectorial de dimensión $n - k$. □

Apliquemos el teorema anterior al código con los parámetros mostrados en el ejemplo 3.1.4

Ejemplo 3.1.6. *Encontrar el límite de Singleton para el código lineal con parámetros $n = 8$ y $d = 3$.*

Solución
De la desigualdad en el teorema 3.1.5 tenemos que la dimensión del código lineal cumple $k \leq n - d + 1$, así tenemos:

$$k \leq 8 - 3 + 1 \Rightarrow k \leq 6$$

Así el límite de Singleton es 6 y de hecho la cantidad de elementos en C es a lo sumo 2^6.

Luego de ver el ejemplo anterior es necesario hacer ciertas observaciones del límite Singleton.

Observaciones:

1. El límite de Singleton es más débil que el límite de Hamming de cierta manera ya que si comparamos el ejemplo 3.1.4 y el 3.1.6 se tiene que para las dimensiones del código lineal se cumple $k \leq 4$ y $k \leq 6$ respectivamente y como vemos el límite de Singleton es más permisible.

2. El teorema 3.1.5 solo da información para un código lineal, en este sentido el límite de Hamming es más débil.

Cuando la igualdad se cumple en el límite de Singleton tenemos un tipo de código útil, con esto tenemos la siguiente definición.

Definición 3.1.7. *Un código lineal (n, k, d) se llamará **código separable de máxima distancia o MDS** por sus siglas en inglés, si $d = n - k + 1$ (o $k = n - d + 1$).*

El siguiente teorema permite caracterizar los códigos MDS de una manera más práctica.

Teorema 3.1.8. *Para C un código lineal (n, k, d), las siguientes afirmaciones son equivalentes.*

1. *$d = n - k + 1$.*

2. *Todos los conjuntos de $n - k$ filas de la matriz de control de paridad son linealmente independiente.*

3. *Todos los conjuntos de k columnas de la matriz generadora son linealmente independientes.*

4. *C es MDS.*

Demostración. Procedamos viendo que 1 es equivalente a 2, 3 y 4 respectivamente, así:

1. Primero veamos que 1 y 2 son equivalentes. Por el teorema 3.1.5 tenemos $d - 1 \leq n - k$, pero si todos los conjuntos de $n - k$ filas de la matriz de control de paridad son l.i. entonces por el teorema 2.9.1 tenemos $d - 1 \geq n - k$ y así tendremos $d - 1 = n - k \Rightarrow d = n - k + 1$. Ahora si $d = n - k + 1$ tendremos que $d - 1 = n - k$ y por el teorema 2.9.1 tenemos que todos los conjuntos de $n - k$ filas de la matriz de control de paridad son l.i.

2. Para ver que 1 y 3 son equivalentes, notemos primero que si $d = n - k + 1$, entonces podemos reescribirlo como $d = n - (k - 1)$, esto quiere decir que el peso mínimo de las palabras de código es $n - (k - 1)$, y por lo tanto la máxima cantidad de ceros en las palabras de código es $k - 1$, así tomemos k columnas de la matriz generadora G de C y formemos la matriz G' con dichas columnas (notemos que la matriz G' es cuadrada de dimensión $k \times k$ ya que la cantidad de filas de G es k por definición y se toman k columnas).

   Ahora sea $u \in K^k$ tal que $uG' = \mathbf{0}$, con u y la matriz generadora G podemos formar la palabra de código $v = uG$, ya que $uG' = \mathbf{0}$ y por como está construida G' tenemos que la palabra v tiene k dígitos iguales a cero o más así $wt(v) \leq n - k$, pero por hipótesis tenemos $d = n - (k - 1)$, esto quiere decir que la palabra de código distinto de cero tiene como máximo $k - 1$ cero, esto implica que $v = \mathbf{0}$ al ser la única palabra de código que tiene más ceros. Por lo tanto como $v = \mathbf{0} \Rightarrow u = \mathbf{0}$, así las filas de G' son l.i. porque $uG' = \mathbf{0}$, para $u = 0$ y no otra palabra distinta de la palabra cero, por lo tanto el rango de la matriz G' es k y así las k columnas de G' son l.i. y por haber tomado de manera arbitrarías las k columnas se tiene que todo conjunto de k columnas de G son l.i.

   Ahora asumamos que todo conjunto de k columnas de G son l.i. así consideremos G' con k columnas de G, y tomemos una palabra $u \in K^k$ tal que $uG' = \mathbf{0}$, pero por hipótesis las k columnas son l.i. el rango de G' es k y por lo tanto las k filas de G' también son l.i. y así $u = \mathbf{0}$ esto implica que la palabra de código que genera u es la palabra cero, así en general siempre que una palabra de código v tenga k o más ceros en sus componentes terminan siendo siempre la palabra cero (ya que al realizar un análisis parecido pero con G' compuesta con las k columnas donde la palabra tiene ceros, se concluye siempre que la palabra de código es la cero). Por lo tanto la única manera para que la palabra de código resultante no sea la palabra cero debe tener $k - 1$ ceros o menos, por lo tanto tenemos que la distancia d de código es $n - (k - 1) \Rightarrow d = n - k + 1$. Así tenemos la equivalencia de 1 y 3.

3. Finalmente 1 y 4 son equivalentes por la definición 3.1.7.

□

Una consecuencia inmediata del anterior teorema es poder concluir que el dual de un código MDS es también MDS.

Corolario 3.1.9. *El dual de un código MDS lineal (n, k, d) es también un código MDS lineal $(n, n - k, k + 1)$.*

Demostración. Sea C un código MDS con parámetros (n, k, d) y sea H la matriz de control de paridad, por definición de distancia todo conjunto de $d - 1$ filas de H son l.i. y por ser C MDS tenemos $d = n - k + 1 \Rightarrow d - 1 = n - k$, por lo tanto tenemos que todos los conjuntos

de $n-k$ filas de H son l.i.

Por otro lado las columnas de H son base para el dual de C, así H^T es la matriz generadora del dual de C y por el párrafo anterior cumple que todos los conjuntos de $n-k$ columnas de H^T son l.i. y por el teorema 3.1.8 tenemos que cumple la 3 y por lo tanto $C^\perp$ es MDS y $d = n-(n-k)+1 = n-n+k+1 = k+1 \Rightarrow d = k+1$. Por lo tanto $C^\perp$ es un código lineal MDS $(n, n-k, k+1)$. □

Un hecho importante en los límites presentados anteriormente es que excluyen algunas posibilidades por ejemplo en los casos 3.1.4 y 3.1.6, el límite de Hamming excluye el valor de $k = 5$ pero el límite de Singleton si lo incluye en las posibilidades para la dimensión de un código lineal C.

De hecho ambos límites solo nos dan un límite superior (por decirlo así) para la cantidad de elementos que puede tener un código pero no nos garantizan si existen todos ellos o no existen, incluso si del rango de posibilidades solo hay uno. Por ello es necesario poder crear un código que tome en cuenta más elementos de los códigos lineales.

Una forma de poder garantizar que un código con parámetros (n, k, d) existe es construir su matriz de control de paridad teniendo en cuenta que la matriz H debe tener todos los conjuntos de $d-1$ filas de H l.i. esto con la idea de que el código creado tenga distancia d. Además las palabras que formarían las filas de H deben de ser de tamaño $n-k$ porque las columnas forman la base del dual del código y si la dimensión del código es k y la del espacio vectorial es n entonces el dual tendrá $n-k$ como dimensión. Con todo esto en mente podemos establecer el último teorema importante de esta sección y que además nos dará una forma de construir la matriz H.

Teorema 3.1.10. ***(Límite de Gilbert-Varshamov)*** *Existe un código lineal de tamaño n, dimensión k y distancia d, si*

$$\binom{n-1}{0} + \binom{n-1}{1} + \cdots + \binom{n-1}{d-2} < 2^{n-k}.$$

Demostración. La idea es construir la matriz de control de paridad para un código lineal con parámetros (n, k, d), las filas son palabras de tamaño $n-k$, así tenemos 2^{n-k} palabras para elegir.

Teniendo en cuenta lo anterior para las primeras $n-k$ filas elegimos palabras con peso igual a uno (esto hace que aparezca la matriz identidad de dimensión $(n-k)\times(n-k)$ y garantiza que las $n-k$ columnas sean l.i.).

Luego para la posición $n-k+1$ no podemos elegir la palabra cero porque ella sería l.d. de las demás que se han elegido. Además no se puede elegir una palabra que sea l.d. de las ya establecidas y como el código debe ser de dimensión d tampoco debe ser combinación

lineal de $d-2$ de ellas (recordemos que $d-1 \leq n-k$ por ello a partir de aquí debemos considerar ese hecho). Así la cantidad de palabras que podemos elegir para esta posición es:

$$2^{n-k} - \left(1 + \binom{n-k}{1} + \binom{n-k}{2} + \cdots + \binom{n-k}{d-2}\right).$$

Al seguir eligiendo de esta manera digamos hasta la fila i-ésima, nos encontraremos con la necesidad de elegir la palabra de tal manera que no sea combinación lineal de $d-2$ palabras ya elegidas (ya que así garantizamos que si tomamos $d-1$ filas serán l.i. porque hemos descartado todas las que son l.d. de $d-2$ de ellas como vimos anteriormente), esto con la idea de que la matriz que estamos construyendo tenga que todo conjunto con $d-1$ filas sean l.i. por ende lo que sucederá en éste caso es que la cantidad de palabras que podemos elegir para la posición i es:

$$2^{n-k} - \left(1 + \binom{i-1}{1} + \binom{i-1}{2} + \cdots + \binom{i-1}{d-2}\right).$$

De igual manera para la posición siguiente tendremos:

$$2^{n-k} - \left(1 + \binom{i}{1} + \binom{i}{2} + \cdots + \binom{i}{d-2}\right)$$

Palabras para elegir y así sucesivamente. Pero esta elección debe parar en algún momento ya que hay un número finito de palabras, además recordemos que por hipótesis tenemos:

$$\binom{n-1}{0} + \binom{n-1}{1} + \cdots + \binom{n-1}{d-2} < 2^{n-k}.$$

De donde:

$$0 < 2^{n-k} - \left(1 + \binom{n-1}{1} + \cdots + \binom{n-1}{d-2}\right).$$

Lo que significa que podemos elegir una palabra más para la posición n de las filas y de hecho esta es la última ya que una matriz de control de paridad de un código (n,k,d) tiene n filas. Con esta última elección lo que resulta es una matriz H de dimensión $n \times (n-k)$ y que todo conjunto de $d-1$ filas de H son l.i. además que sus columnas son l.i. por lo tanto es la matriz de control de paridad de un algún código lineal C con parámetros (n,k,d); en otras palabras hemos creado un código lineal ya que a partir de los algoritmos vistos en el capítulo 2 podemos encontrar la matriz generadora de C. □

Un resultado que es consecuencia del límite de Gilbert-Varshamov es el siguiente:

Corolario 3.1.11. *Si $n \neq 1$ y $d \neq 1$, entonces existe un código lineal C de tamaño n y distancia al menos d con:*

$$|C| \geq \frac{2^{n-1}}{\binom{n-1}{0} + \binom{n-1}{1} + \cdots + \binom{n-1}{d-2}}.$$

Demostración. Sean n y d tal que $d \leq n$ y $d > 2$, entonces tenemos que se cumple:

$$\binom{n-1}{0} + \binom{n-1}{1} + \cdots + \binom{n-1}{d-2} < \binom{n-1}{0} + \binom{n-1}{1} + \cdots + \binom{n-1}{n-1}.$$

El lado derecho de la desigualdad es igual a 2^{n-1}, así tendremos:

$$\binom{n-1}{0} + \binom{n-1}{1} + \cdots + \binom{n-1}{d-2} < 2^{n-1}.$$

Ahora tomando $k = 1$ podemos aplicar el teorema 3.1.10, y dicho teorema nos garantiza que existe un código lineal C_1 con parámetros $(n, 1, d)$. Ahora tenemos dos casos:

Caso I:

$$|C_1| \left[\binom{n-1}{0} + \binom{n-1}{1} + \cdots + \binom{n-1}{d-2} \right] < 2^{n-1}.$$

Caso II:

$$|C_1| \left[\binom{n-1}{0} + \binom{n-1}{1} + \cdots + \binom{n-1}{d-2} \right] \geq 2^{n-1}.$$

Si sucede el caso II ya se tiene el resultado. Pero si sucede el caso I tenemos:

$$|C_1| \left[\binom{n-1}{0} + \binom{n-1}{1} + \cdots + \binom{n-1}{d-2} \right] < 2^{n-1}.$$

Pero como $|C_1| = 2^1$, al sustituir tendremos:

$$2 \left[\binom{n-1}{0} + \binom{n-1}{1} + \cdots + \binom{n-1}{d-2} \right] < 2^{n-1}.$$

De donde obtendremos:

$$\left[\binom{n-1}{0} + \binom{n-1}{1} + \cdots + \binom{n-1}{d-2} \right] < 2^{n-2}.$$

Como vemos al tomar $k = 2$ y la desigualdad anterior por el teorema 3.1.10, existe un código lineal C_2 con parámetros $(n, 2, d)$. Con este nuevo código volvemos a tener dos desigualdades, o sucede:

$$|C_2| \left[\binom{n-1}{0} + \binom{n-1}{1} + \cdots + \binom{n-1}{d-2} \right] < 2^{n-1}.$$

O puede pasar:

$$|C_2|\left[\binom{n-1}{0}+\binom{n-1}{1}+\cdots+\binom{n-1}{d-2}\right]\geq 2^{n-1}.$$

Pero si sucede esto último tenemos el resultado, pero si en cambio sucede la otra, entonces teniendo en cuenta que $|C_2| = 2^2$ y repitiendo los despejes hechos para C_1 tendremos la desigualdad:

$$\left[\binom{n-1}{0}+\binom{n-1}{1}+\cdots+\binom{n-1}{d-2}\right]<2^{n-3}.$$

Así al tomar $k = 3$ y la desigualdad anterior podemos concluir por el teorema 3.1.10 que existe un código lineal C_3 con parámetros $(n, 3, d)$. Eventualmente debido a que 2^{n-1} es finito encontraremos un código lineal C_k con parámetros (n, k, d) que cumpla:

$$|C_k|\left[\binom{n-1}{0}+\binom{n-1}{1}+\cdots+\binom{n-1}{d-2}\right]\geq 2^{n-1}.$$

Dicho código es el buscado ya que de la desigualdad anterior al despejar tendremos:

$$|C_k|\geq \frac{2^{n-1}}{\binom{n-1}{0}+\binom{n-1}{1}+\cdots+\binom{n-1}{d-2}}.$$

Con esto se completa la prueba ya que hemos encontrado un código lineal con distancia igual a d y además cumple la desigualdad deseada con $n \neq 1$ y $d \neq 1$. □

Observaciones: El teorema 3.1.10 nos da una forma de como construir una matriz de control de paridad para encontrar un código lineal C con distancia deseada, además el teorema nos da una relación entre la cantidad de elementos de un código y sus parámetros (n, k, d):

1. Lo más destacable y útil del teorema es poder decirnos si un código existe (pero no, si no existe); ya que de antemano nosotros podemos saber si nuestras exigencias entorno al código son razonables (Recordemos que saber si un código puede detectar o corregir cierta cantidad de errores está ligado a la distancia que este posee).

2. Lo segundo es que el teorema también nos da un límite en torno a la cantidad de elementos que puede poseer el código lineal, ya que al reescribir adecuada mente la desigualdad tendremos:

$$|C|<\frac{2^n}{\binom{n-1}{0}+\binom{n-1}{1}+\cdots+\binom{n-1}{d-2}}, \text{ donde } |C| = 2^k.$$

También es necesario denotar que el corolario 3.1.11 es un límite muy especial a diferencia de los anteriores ya que todos los límites mostrados el de Hamming, Singleton y Gilbert-Varshamov dan una cota superior a la cantidad de elementos de un código lineal (el caso de Hamming puede ser no lineal), pero en el caso del corolario nos ofrece una cota

inferior para la cantidad de elementos del código lineal. Así al combinarlo con los límites anteriores llegamos a precisar mejor la cantidad de palabras que debe poseer un código lineal.

Para terminar la sección veamos dos ejemplos:

Ejemplo 3.1.12. *¿Existe un código lineal de tamaño $n = 9$, dimensión $k = 2$ y distancia $d = 5$?*

Solución
Para determinar la existencia apliquemos el teorema 3.1.10, así primero calculemos los combinatorios:

$$\binom{8}{0} + \binom{8}{1} + \binom{8}{2} + \binom{8}{3} = 93.$$

Por otro lado $2^{9-2} = 2^7 = 128$. Como vemos $93 < 128$, por lo tanto existe un código lineal $(9, 2, 5)$.

Ejemplo 3.1.13. *Calcular una cota interior y superior para la dimensión k de un código lineal $(9, k, 5)$.*

Solución
Recordemos que la cardinalidad de un código lineal es 2^k, donde k es la dimensión, así solo debemos saber como está acotado inferior y superiormente la cardinalidad del código para saber también el intervalo de k.

Primero veamos la cota inferior, para ello usamos el corolario 3.1.11:

$$|C| \geq \frac{2^{9-1}}{\binom{8}{0} + \binom{8}{1} + \binom{8}{2} + \binom{8}{3}} = \frac{2^8}{93} = \frac{256}{93} \approx 2{,}75.$$

Como deseamos un código lineal debemos ver la menor potencia de dos que sea mayor que 2,75, la cual es 2^2, así $|C| \geq 2^2$. Ahora para la cota superior usaremos el límite de Hamming:

$$|C| \leq \frac{2^9}{\binom{9}{0} + \binom{9}{1} + \binom{9}{2}} = \frac{512}{46} \approx 11{,}13.$$

De igual forma al buscar la mayor potencia de 2 que verifique ser menor o igual que 11,13 es 2^3, por lo tanto $|C| \leq 2^3$. Al combinar ambos resultados tenemos:

$$2^2 \leq |C| \leq 2^3 \Rightarrow 2 \leq k \leq 3.$$

Así existen códigos lineales $(9, 2, 5)$ y $(9, 3, 5)$.

3.2. Códigos perfectos

En esta sección presentamos los códigos perfectos, dando su definición y ejemplos. Las secciones finales las dedicaremos a exponer dos tipos de códigos perfectos así como algunos códigos asociados a ellos.

Definición 3.2.1. *Un código lineal C con parámetros $(n, k, 2t+1)$ se llamará **código perfecto**, si $|C|$ es igual al límite de Hamming.*

Observación: De la definición es fácil ver que lo que buscamos es que un código lineal C cumpla:

$$|C| = \frac{2^n}{\binom{n}{0} + \binom{n}{1} + \cdots + \binom{n}{t}}.$$

Y es precisamente al hecho que dividimos por $\binom{n}{0} + \binom{n}{1} + \cdots + \binom{n}{t}$ es que es difícil encontrar un código perfecto ya que debemos buscar un valor de t tal que la división sea entera y potencia de dos; ya que siempre deseamos encontrar un código lineal.

Veamos ahora algunos ejemplos de códigos lineales que son perfectos:

Ejemplo 3.2.2. *Si $t = 0$ encontrar el código lineal perfecto $(n, k, 1)$.*

Solución
Como deseamos tener un código perfecto apliquemos la definición y veamos que cardinalidad debería tener dicho código:

$$|C| = \frac{2^n}{\binom{n}{0}} = \frac{2^n}{1} = 2^n.$$

Así el único código lineal que tiene 2^n palabras es K^n, por lo tanto el código lineal $C = K^n$ es el código lineal perfecto buscado.

Ejemplo 3.2.3. *Si $n = d = 2t+1$. Encontrar el código lineal $(2t+1, k, 2t+1)$.*

Solución
Antes que nada recordemos una propiedad de los combinatorios:

$$\binom{n}{i} = \binom{n}{n-i}.$$

Ahora como $n = 2+1$, tenemos para $i = t$:

$$\binom{n}{t} = \binom{n}{n-t} = \binom{n}{2t+1-t} = \binom{n}{t+1}.$$

Con todo lo anterior (de hecho la última igualdad nos muestra para i se llega a la mitad de los combinatorios) y el hecho que n es impar podemos tener:

$$\binom{n}{0} + \binom{n}{1} + \cdots + \binom{n}{t} = \frac{1}{2}\left(\binom{n}{0} + \binom{n}{1} + \cdots + \binom{n}{n}\right) = \frac{1}{2} * 2^n = 2^{n-1}.$$

Con esto podemos establecer:

$$|C| = \frac{2^n}{\binom{n}{0} + \binom{n}{1} + \cdots + \binom{n}{t}} = \frac{2^n}{2^{n-1}} = 2$$

El único código lineal que cumple con tener dos palabras y tener distancia $2t+1$, es el código formado por la palabra cero y la palabra con todos sus dígitos iguales a uno.

Los dos ejemplos anteriores nos dan dos tipos de códigos perfectos muy especiales, estos códigos son llamados **códigos perfectos triviales**.

Los códigos perfectos triviales no son los únicos que existen como veremos en el siguiente ejemplo:

Ejemplo 3.2.4. *Sea $n = 23$ y $d = 7$, verificar si existe un código perfecto con estos parámetros.*

Solución
Para saber si existe comprobamos si al aplicar la definición la cardinalidad es una potencia de dos (solo basta verificar esto ya que la distancia ya es un número impar). Como $d = 7$ tenemos que $t = 3$, así:

$$|C| = \frac{2^{23}}{\binom{23}{0} + \binom{23}{1} + \binom{23}{2} + \binom{23}{3}} = \frac{2^{23}}{2048} = \frac{2^{23}}{2^{11}} = 2^{12} = 4096 \text{ palabras.}$$

Esto muestra que un código perfecto de tamaño $n = 23$ y distancia $d = 7$ existe. Este tipo de código es llamado código de Golay lo estudiaremos al final de este capítulo.

Como hemos visto existen los códigos perfectos, pero aun así nos gustaría saber: ¿quienes son? o al menos ¿cuáles son sus parámetros? Este problema fue resuelto en el sentido de dar una estructura a los parámetros de un código perfecto y dicho resultado fue establecido por **Tietäväiren** y **Van Lint** en 1973; lamentablemente no será probado en este documento ya que requiere un estudio profundo de polinomios en K, además de introducir las nociones de teoría de grupo para poder establecerlo y es algo que se escapa de nuestras manos con las herramientas ya establecidas.

Teorema 3.2.5. *Si C es un código perfecto no trivial de tamaño n y distancia $d = 2t+1$, entonces $n = 23$ y $d = 7$, o $n = 2^r - 1$ para algún $r \geq 2$ y $d = 3$.*

Por último veamos un teorema que expresa una característica particular de los códigos perfectos:

Teorema 3.2.6. *Si C es un código perfecto de tamaño n y distancia $d = 2t + 1$, entonces C corregirá todos los patrones de error de peso menor o igual que t y ningún otro patrón.*

Demostración. Como la distancia del código lineal es $d = 2t + 1$ tenemos $t = (d - 1)/2$, entonces por el teorema 1.8.6 tenemos que C corregirá todos los patrones de error de peso menor o igual a t. Además por lo visto en el capítulo 2 sabemos que estos serán los líderes de sus clases laterales. Ahora por la definición 3.1.1 sabemos que hay $\binom{n}{0} + \binom{n}{1} + \cdots + \binom{n}{t}$ patrones de error de peso menor o igual a t.

Además como C es perfecto y usando el teorema 2.7.3, tenemos:

$$|C| = \frac{2^n}{\binom{n}{0} + \binom{n}{1} + \cdots + \binom{n}{t}} \Rightarrow \binom{n}{0} + \binom{n}{1} + \cdots + \binom{n}{t} = \frac{2^n}{|C|}.$$

Por lo tanto la cantidad de clases laterales es:

$$\binom{n}{0} + \binom{n}{1} + \cdots + \binom{n}{t}.$$

Esto significa que el número de las clases laterales es igual al número de patrones de error que son los líderes en sus clases, que no hay más patrones de error. Por lo tanto C corrige todos los patrones de error de peso menor o igual que t y ningún otro. □

La propiedad que describe el teorema 3.2.6 es sumamente valiosa ya que quiere decir que todas las palabras (no del código) de K^n están a una distancia máxima t de una palabra de código de C y esto los hace perfecto al código ya que es capaz de reunir las palabras que no están en el código en bolas (como la noción de espacios métricos y sus abiertos). Así reuniendo todas las posibles fallas de cada palabra de código.

Por último al corregir todos los patrones de error de peso menor o igual que t, un código perfecto es llamado también **código perfecto lineal t-corrector**.

3.3. Más códigos perfectos - Códigos de Hamming

En esta sección definiremos un tipo de código perfecto que es muy fácil de utilizar y además de ser un código 1-corrector de errores. Aunque solo nos corrige un error, la importancia de estos códigos radica en que son fáciles de usar en la codificación y decodificación.

Definición 3.3.1. *Un código de tamaño $2^r - 1, r \geq 2$, que tiene una matriz H de control de paridad tal que sus filas son todas las palabras de tamaño r distintas de la palabra cero, es llamado **código de Hamming** de tamaño $2^r - 1$.*

Veamos algunas propiedades importantes de los códigos de Hamming:

Teorema 3.3.2. *Un código C de Hamming tiene* 2^{2^r-1-r} *palabras.*

Demostración. De la definición 3.3.1 tenemos que la matriz H tiene las palabras de peso igual a 1 y de tamaño r, así las columnas de la matriz H son l.i. entonces el código de Hamming tiene dimensión $2^r - 1 - r$; recordemos que la dimensión del dual es $n - k$ que es igual a la cantidad de columnas que tienen H y por lo tanto si quiero saber la dimensión del código simplemente a $n = 2^r - 1$ le resto la cantidad de columnas de H. Por lo tanto el código de Hamming tiene 2^{2^r-1-r} palabras. □

Teorema 3.3.3. *La distancia de un código C de Hamming es* $d = 3$.

Demostración. Como la matriz H de la definición 3.3.1 tiene todas las palabras de tamaño r distintas de la palabra cero entonces al tomar dos palabras estas son l.i. ya que la única manera de que den la palabra cero es que sean la misma pero ninguna se repite en las filas de H. Pero al tomar tres palabras, por ejemplo $100\ldots00$, $010\ldots00$ y $110\ldots00$ (ya que H tiene todas las palabras distintas de cero) estas palabras son l.d. claramente entonces por lo visto en el capítulo 2 tenemos que la distancia del código de Hamming es $d = 3$. □

Teorema 3.3.4. *Un código lineal de Hamming es perfecto.*

Demostración. Para probar que es un código de Hamming bastará demostrar que $|C|$ es igual al límite de Hamming ya que $n = 2^r - 1$ y por el teorema 3.2.5 es candidato a ser perfecto.

Por 3.3.3 tenemos que $d = 3$, así $t = 1$, por lo tanto tendremos:

$$\frac{2^n}{\binom{n}{0} + \binom{n}{1}} = \frac{2^n}{1+n}.$$

Pero $n = 2^r - 1$, así tendremos:

$$\frac{2^n}{1+n} = \frac{2^{2^r-1}}{1+2^r-1} = \frac{2^{2^r-1}}{2^r} = 2^{2^r-1-r}$$

Y por 3.3.2 y lo anterior tenemos:

$$|C| = \frac{2^n}{\binom{n}{0} + \binom{n}{1}}.$$

Por lo tanto es un código perfecto. □

Por este último teorema un código de Hamming también es llamado **código de Hamming corrector de errores simples**.

Veamos un ejemplo de un código de Hamming.

Ejemplo 3.3.5. *Una matriz de control de paridad para un código de Hamming de parámetros* $(2^3 - 1, 4, 3) = (7, 4, 3)$ *es:*

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Usando el algoritmo presentado en el capítulo 2 tenemos que la matriz generadora del código de Hamming es:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}$$

Basta ver este ejemplo porque todos los demás códigos que tienen los mismo parámetros son equivalentes (esto por la definición de código equivalente en el capítulo 2).

Por último es sumamente fácil construir la matriz de codificación estándar (SDA por sus siglas en ingles). Ya que como corrige los patrones de error de peso uno y cero al ser perfecto y por el teorema 3.2.6 estos son los únicos líderes en las clases laterales, entonces tendremos que en el caso de ser de peso uno el patrón de error solo darán como síndrome una fila de H que corresponde al lugar donde está el problema. Por lo tanto la SDA tiene la forma:

Representantes de las clases laterales	Síndrome
00...00	0...0
I_{2^r-1}	H

Como vemos es sumamente fácil y práctico trabajar con los códigos de Hamming ya que al solo suceder un error tendremos que ver en qué posición se encuentra el síndrome y luego a partir de ahí sabremos que palabra de la matriz identidad es nuestro patrón de error para finalmente recuperar la palabra enviada.

Aunque son muy útiles los códigos de Hamming al solo corregir un error no los hace muy prácticos para canales con un ruido que afecta más de una vez cada palabra. Por ellos

terminaremos nuestro estudio de los códigos perfectos con los códigos de Golay ya que estos brindarán un mayor rango de corrección.

3.4. Códigos extendidos

En las primeras secciones del capítulo 1 estudiamos como al aumentar la cantidad de dígitos a las palabras del código hace que el nuevo código tenga propiedades mejoradas a diferencia del código inicial (ya sea para detectar o corregir errores), estudiaremos en esta sección el método del bit de paridad como se hizo en el capítulo 1, así centraremos nuestra atención a un código aumentado en un dígito nada más, sin embargo esta práctica de aumentar dígitos puede hacerse aumentando más de un dígito en las palabras del código a costa de una tasa de información baja.

Definición 3.4.1. *Sea C un código lineal de tamaño n. El código C^* de tamaño $n+1$ obtenido de C al agregar un dígito a cada palabra de código de tal manera que todas las palabras de código tengan un peso par es llamado* ***código extendido de*** *C.*

Consideremos ahora la matriz generadora de tamaño $k \times n$ de un código lineal C, entonces la matriz generadora G^* de tamaño $k \times (n+1)$ del código extendido de C tal que:

$$G^* = [G, b].$$

Donde b es la última columna de G^* de tal manera que las filas de G^* tienen peso par.

Ya con la matriz generada para G^* es fácil encontrar la matriz control de paridad H usando el algoritmo del capítulo 2, pero hay una forma más fácil de calcularla, así tenemos:

Teorema 3.4.2. *Si G^* es la matriz generadora del código extendido C^* donde $G^* = [G, b]$ y H la matriz de control de paridad de G, entonces la matriz de control de paridad para G^* es:*

$$H^* = \begin{bmatrix} H & j \\ \boldsymbol{0} & 1 \end{bmatrix}.$$

Donde j es un matriz columna de tamaño $n \times 1$ de unos y $\boldsymbol{0}$ es la palabra cero de tamaño $n-k$. Además H^* es de dimensión $(n+1) \times (n-k+1)$.

Demostración. Primero notemos que el rango de H es $n-k$ y por construcción la columna:

$$\begin{bmatrix} j \\ 1 \end{bmatrix}$$

Es una columna linealmente independiente a las de:

$$\begin{bmatrix} H \\ \mathbf{0} \end{bmatrix}$$

Ya que se agregan al final ceros a cada columna de H y la última fila tiene un uno al final haciendo que las $n-k+1$ columnas de H^* sean l.i. así H^* tiene rango $n-k+1$.

Por otro lado verifiquemos que $G^*H^* = \mathbf{0}$:

$$\begin{aligned} G^*H^* &= [G, b] \begin{bmatrix} H & j \\ \mathbf{0} & 1 \end{bmatrix} \\ &= [GH, Gj + b]. \end{aligned}$$

Pero por definición $GH = \mathbf{0}$, además al ser j una palabra llena de unos el producto Gj es la suma de los dígitos de cada palabra de G y como b es la palabra columna de tal manera que el peso de las filas de G sean de peso par tenemos entonces $Gj + b = \mathbf{0}$. Por lo tanto:

$$G^*H^* = [GH, Gj + b] = [\mathbf{0}, \mathbf{0}] = \mathbf{0}.$$

Así concluimos H^* anula a G^* (y por lo tanto a todas las palabras que genera G las anula también) y como sus columnas son linealmente independientes entonces es una matriz de control de paridad para C^* como se deseaba verificar. □

Luego de establecer esta manera práctica para encontrar la matriz de control de paridad para un código extendido lo aplicamos a un caso en concreto.

Ejemplo 3.4.3. *Considere el código C con matriz generadora G y matriz de control de paridad H tal que:*

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

Y

$$H = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

Calcular la matriz generadora y matriz de control de paridad para el código extendido C^*.

Solución
Usando lo expuesto antes tenemos:

$$G^* = \left[\begin{array}{ccccc|c} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 \end{array}\right].$$

Y

$$H^* = \left[\begin{array}{cc|c} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \\ \hline 0 & 0 & 1 \end{array}\right].$$

Donde G^* y H^* son la matriz generadora y matriz de control de paridad respectivamente para C^*.

Como vemos ya obtenida la matriz generadora y la de control de paridad es sumamente fácil encontrar las mismas matrices para el código lineal C^* (código extendido). Veamos ahora como el hecho de agregar un solo dígito a nuestro código mejora considerablemente el código lineal en el sentido de detección de errores.

Teorema 3.4.4. *Sea un código lineal C con distancia d, entonces el código extendido C^* tienen distancia $d^* = d + 1$, si d es impar y $d^* = d$, si d es par.*

Demostración. Analicemos por casos:

1. Si d es impar, entonces $\forall v \in C$, $wt(v) \geq d$. En el caso que el peso sea par entonces tendremos que la palabra $v0 \in C^*$ tiene peso $w(v0) = wt(v) \geq d + 1$ (ya que al ser par y ser mayor que d debe ser al menos igual $d + 1$). Por otro lado si el peso es impar entonces $v1 \in C^*$ tendrá peso par tal que $wt(v1) = wt(v) + 1 \geq d + 1$, de hecho al hacer un análisis parecido a la palabra $w \in C$ tal que $wt(w) = d$ se concluye que $wt(w1) = d + 1$. Por lo tanto la distancia de C^* es $d^* = d + 1$.

2. Si d es par, entonces $\forall v \in C$, $wt(v) \geq d$, pero si palabra es de peso impar tendremos $wt(v1) = wt(v) + 1 > d + 1 > d \Rightarrow wt(v) > d$. Ahora si es de peso par no habrá cambio $wt(v0) \geq d$ de hecho en la palabra $w \in C$ tal que $wt(w0) = d$; por tener peso par. Por lo tanto la distancia de C^* es $d^* = d$.

Así conseguimos en ambos caso lo deseado. □

Observaciones: El teorema 3.4.4 nos da dos moralejas, la primera de ellas es que es mejor agregar el bit de paridad para códigos de distancia d impar pues su distancia aumenta (no así si, si la distancia es par en dicho caso queda la misma distancia) y por ende aumenta la cantidad de patrones que logra detectar (esto por lo demostrado en el teorema 1.7.8) y además en algunos casos podrá tener patrones de error extras que también podrá detectarlo (por el teorema 1.8.6, ya que al tomar el menor entero puede que aumente o no la estimación para el peso de los patrones de error a detectar).

La segunda moraleja es que no tiene sentido aplicar dos veces el método del bit de paridad pues al aplicarlo una vez tendremos un código de distancia par en ambos casos y al volver aplicar debido a que siempre tenemos distancia par, entonces el resultado será otro código extendido con mayor tamaño pero con la misma distancia; debido a este fenómeno que ya no vuelve a cambiar la distancia es que será inútil volver agregar el bit de paridad las palabras del código extendido C^*.

En la siguiente sección veremos una aplicación muy importante al método de agregar el bit de paridad a un código perfecto.

3.5. Código extendido de Golay

En esta sección y la siguiente discutiremos el último de los dos tipos de códigos perfectos. De hecho primero estudiaremos la extensión del código de Golay para luego ver propiamente los códigos de Golay este cambio de lógica de presentación es porque es más fácil implementar el algoritmo de corrección para los códigos de Golay a partir del código extendido. Así sea la matriz B de 12×12, tal que:

$$B = \begin{bmatrix} 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \end{bmatrix}$$

Con B definamos nuestra matriz generadora G de 12×24, de la siguiente manera:

$$G = [I, B]$$

Donde I es la matriz identidad de 12×12. El código lineal C que genera la matriz G es llamado **código extendido de Golay** y lo denotaremos por C_{24}. Notemos también que C_{24} es un *código sistemático* ya que tiene su matriz generadora en forma estándar.

Por como hemos definido la matriz B es simétrica (esto se puede corroborar por simple inspección) en otras palabras $B^T = B$. Además B cumple $BB = \mathbf{0}$; esto también se puede ver por simple inspección que al multiplicarla por sigo mismo da la matriz cero de hecho

cada fila es ortogonal a las demás como se ve al tomar cualquieras dos filas y al hacer el producto punto estos darán cero; por ejemplo $110111000101 \cdot 101110001011 = 0$, por esto es que $BB = \mathbf{0}$.

Teorema 3.5.1. *El código lineal C_{24}, con matriz generadora $G = [I, B]$ cumple:*

1. *La dimensión de C_{24} es $k = 12$ y $n = 24$.*
2. *Una matriz de control de paridad para C_{24} de dimensión 24×12 es:*
$$H = \begin{bmatrix} B \\ I \end{bmatrix}.$$
3. *Otra matriz de control de paridad para C_{24} de dimensión 24×12 es:*
$$H = \begin{bmatrix} I \\ B \end{bmatrix}.$$
4. *Otra matriz generadora para C_{24} de 12×24 es $[B, I]$.*
5. *C_{24} es auto-dual ($C_{24} = C_{24}^{\perp}$).*
6. *La distancia de C_{24} es $d = 8$.*
7. *C_{24} es un código tres-corrector de errores.*

Demostración. Demostremos de manera ordenada cada numeral.

1. Como la matriz generadora de C_{24} tiene dimensión 12×24, entonces la dimensión del código es claro que es $k = 12$ y el tamaño de las palabras es $n = 24$.
2. Ya que G está en forma estándar entonces al aplicar el algoritmo 2.3.3 y tendremos que H es:
$$H = \begin{bmatrix} B \\ I \end{bmatrix}.$$
3. Para saber si la matriz es una matriz de control de paridad esta debe tener sus columnas l.i. y además anular a la base del código lineal en el caso de:
$$H = \begin{bmatrix} I \\ B \end{bmatrix}.$$
Ya que tiene la matriz identidad sus columnas son l.i., basta con verificar si anula a G y además debemos recordar que $B^2 = I$, así:

$$GH = [I, B] \begin{bmatrix} I \\ B \end{bmatrix} = I^2 + B^2 = I + I = \mathbf{0}.$$

Así $GH = \mathbf{0}$. Por lo tanto la matriz H cumple ser la matriz de control de paridad para el código C_{24}.

4. Para verificar que la matriz $[B, I]$ es otra matriz generadora para el código, basta ver que se anula con cualquiera de las matrices de control de paridad ya que cumple que tiene 12 filas l.i. por tener la matriz identidad en su definición hace que cada fila se linealmente independiente. Así:

$$[B, I] \begin{bmatrix} I \\ B \end{bmatrix} = B + B = \mathbf{0}.$$

Por lo tanto se anula con la matriz de control de paridad así $[B, I]$ es otra matriz generador para el código C_{24}.

5. Primero notemos que H la matriz de control de paridad definida en 2 tiene como columnas la base para el dual del código, esto quiere decir que H^T es la matriz generadora para $C_{24}^{\perp}$, donde $H^T = [B, I]$. Pero al usar el resultado anterior $[B, I]$ cumple ser también la matriz generadora de C_{24}, de esto se deduce que genera aún mismo código y por lo tanto $C_{24} = C_{24}^{\perp}$.

6. Para demostrar que la distancia del código C_{24} es ocho, veamos por parte:

Parte I Primero veamos que $\forall v \in C_{24}, wt(v)$ es múltiplo de cuatro. Así notemos primero que las filas de G tiene peso 8 o 12. Luego tomemos una palabra v tal que sea formada por dos filas de G digamos r_i y r_j, o sea $v = r_i + r_j$. Debido a que las filas de B son ortogonales entre si, también lo serán las filas de G, por lo tanto $r_i \cdot r_j = 0$, esto quiere decir que tienen un número par de unos en común digamos $2x$, así el peso de v se puede expresar:

$$wt(v) = wt(r_i) + wt(r_j) - 2(2x) = wt(r_i) + wt(r_j) - 4x.$$

Pero las dos filas o bien su peso es 8 o 12 pero cualquier caso son múltiplo de cuatro. por lo tanto la resta es múltiplo de cuatro, así el peso de v es múltiplo de cuatro. Ahora si suponemos que v es una palabra que se forma por la suma de tres filas tendremos:

$$v = r_i + r_j + r_l.$$

Luego sea $w = r_i + r_j$ y así tendremos que el peso de w por lo demostrado antes que tiene peso múltiplo de cuatro, finalmente repitiendo el análisis anterior ya que r_l es ortogonal a w ya que es la suma de dos palabras ortogonales a él, se podrá concluir

que el peso de v es múltiplo de cuatro. Al ir aplicando así sucesivamente el análisis tendremos que cualquier suma de las filas de G se tiene que son múltiplos de cuatro.

Parte II Debemos notar que las primeras once filas de G tienen peso 8, esto quiere decir que la distancia del código debe ser o bien 4 u 8; ya que lo demostrado en **I** verifica que todas las palabras distintas a la palabra cero de C_{24} son múltiplo de cuatro y hemos encontrado al menos una palabra de peso 8 entonces lo mínimo puede ser 4 o 8.

Parte III Veamos que no hay palabras de peso igual a 4 en el código. Así supongamos que existe v en C_{24} tal que $wt(v) = 4$. Usando las matrices generadoras usadas en los numerales 1 y 4, existen u_1 y u_2 de K^{12} entonces tenemos que v se puede expresar como:

$$v = [u_1, u_1B] \text{ y } v = [u_2B, u_2].$$

Por lo tanto el peso de v es $wt(v) = wt(u_i) + wt(u_iB)$ (para ambos casos). Además como $wt(v) = 4$, entonces tenemos que $wt(u_1) \leq 2$ o $wt(u_2) \leq 2$ (Ya que a lo sumo dos dígitos iguales a uno están en los primeros 12 dígitos o en los 12 últimos). Ahora veamos por caso:

Si $wt(u_1) = 0$ entonces $wt(v) = 0$ y esto es una contradicción ya que el peso de v es cuatro.

Si $wt(u_1) = 1$ entonces $wt(v) = 1 + wt(u_1B) \Rightarrow wt(u_1B) = 3$ pero esto es imposible porque al tener peso uno u_1, solo toma una fila de B cuando se hace el producto u_1B y una fila de B tiene peso 7 o 11 pero no 3.

Si $wt(u_1) = 2$ entonces $wt(v) = 2 + wt(u_1B) \Rightarrow wt(u_1B) = 2$, esto también es imposible ya que al tener peso 2 la palabra u_1 quiere decir que suma dos filas cuando se hace el producto u_1B y por demostrado en **I** la suma de dos filas de G puede ser múltiplo de cuatro, esto implica que la suma de dos filas de B tienen peso como mínimo 3; porque en el caso extremo la suma da 4 en las filas de G y para regresar a las palabras de B debemos quitar los primeros 12 dígitos de G pero eso solo es restar un uno en la cantidad de unos que tenia. Por lo tanto no puede ser que u_1B tenga peso igual a dos.

Es el mismo análisis para u_2, así en ningún caso es posible por lo tanto no existe palabra de código con peso igual a 4 y por lo tanto el mínimo peso es 8 y así la distancia del código es $d = 8$.

7. Ya que por el numeral anterior la dimensión del código es $d = 8$ entonces al aplicar el teorema 1.8.6 y tendremos que C_{24} corrige todos los patrones de peso menor o igual a 3 (ya que $\lfloor (d-1)/2 \rfloor = \lfloor (8-1)/2 \rfloor = \lfloor 7/2 \rfloor = \lfloor 3{,}5 \rfloor = 3$). Por lo tanto C_{24} es un código tres-corrector de errores.

Así el teorema queda demostrado. □

Con la demostración del teorema anterior estamos listos para crear un algoritmo que nos permita poder corregir errores de peso a lo sumo 3 para el código C_{24}.

3.5.1. Corrección de errores para el código C_{24}

En este último apartado nos dedicaremos a crear un algoritmo que nos permita calcular el líder la clase lateral donde está la palabra recibida sin tener que hacer una tabla para deducir donde ocurre el error. Para ellos nos referiremos a la palabra recibida como w y la palabra más cercana a w del código como v, además el patrón de error es $u = w + v$, también separaremos por una coma los primeros 12 dígitos de los 24, y también el patrón de error u lo denotaremos como $u = [u_1, u_2]$, donde u_1, u_2 son palabras de tamaño 12.

Por el numeral 7 del teorema 3.5.1 el código C_{24} corrige todos los patrones de peso a lo sumo 3, así supongamos que $wt(u) \leq 3$. Como $u = [u_1, u_2]$, entonces tendremos que $wt(u_1) \leq 1$ o $wt(u_2) \leq 1$. Ahora calculemos el síndrome s_1 de $w = v + u$ usando la matriz de control de paridad:

$$H = \begin{bmatrix} I \\ B \end{bmatrix}.$$

Así tendremos $s_1 = wH = uH = u_1 + u_2B \Rightarrow s_1 = u_1 + u_2B$. Ahora si $wt(u_2) = 0$ entonces $s_1 = u_1$, esto quiere decir que el peso de s_1 es a lo sumo 3 y el error ocurre en los primeros doce dígitos. Por otro lado si $wt(u_2) = 1$, entonces tendremos que $s_1 = u_1 + b_i$, donde b_i es la fila que corresponde a la posición donde ocurre el error en el segundo bloque de dígitos; esto es así ya que como u_2 tiene peso igual a uno quiere decir que el producto u_2B toma solo una fila de B y dicha fila va en la posición de ocurre el error. Además ya que $s_1 = u_1 + b_i \Rightarrow s_1 + b_i = u_1$, entonces la palabra $s_2 + b_i$ tiene peso a lo sumo dos (y de hecho da la posición de los dos dígitos cambiados) ya que el tercer error ocurre en el segundo bloque de los 12 dígitos ($wt(u_2) = 1$).

Ahora si calculamos el segundo síndrome usando ahora la matriz de control de paridad:

$$H = \begin{bmatrix} B \\ I \end{bmatrix}.$$

Tendremos que $s_2 = wH = u_1B + u_2$, y de igual manera se llegan a conclusiones similares para $w(u_1) \leq 1$ pero para los casos restantes (las otras posibilidades cuando en el segundo bloque de dígitos ocurre 3 errores, 2 errores y un error pero ninguno en el primer bloque, así como cuando ocurre uno en el primer bloque y dos en el segundo).

Como vemos en el análisis hecho anteriormente si ocurren como máximo tres errores es posible elegir adecuadamente las filas correspondientes en cualquiera de las dos matrices de control de paridad para formar el patrón de error correcto y poder conocer la palabra enviada, ya que al tener w y construir u, seremos capaces de encontrar $v = w + u$.

Así lo que hemos creado es un algoritmo que nos permite corregir como máximo tres errores. Pero antes de establecer adecuadamente el algoritmo recordemos que $B^2 = I$ y además establezcamos una relación entre los síndromes s_1 y s_2 que será de gran utilidad:

$$\begin{aligned} s_1 &= u_1 + u_2B = wH \\ s_2 &= u_1B + u_2 \\ &= u_1B + u_2I \\ &= u_1B + u_2B^2 \\ &= (u_1 + u_2B)B \\ &= wHB \\ &= s_1B. \end{aligned}$$

Por lo tanto $s_2 = s_1B$ y gracias a esta igualdad no hace falta de implementar ambas matrices de control de paridad, así solo utilizaremos en el algoritmo:

$$H = \begin{bmatrix} I \\ B \end{bmatrix}.$$

Finalmente denotemos por e_i como la palabra de tamaño 12 donde tiene un dígito igual a uno en la posición i y ceros en las demás. También b_i es la fila de B en la posición i (la posición i representa el lugar donde ocurre el error). Así finalmente tendremos el siguiente algoritmo usando el enfoque IMLD:

Algoritmo 3.5.2. ***(IMLD para*** C_{24}***)*** *Sea w una palabra recibida para calcular el patrón de error u procedemos de la siguiente manera:*

1. *Calculamos el síndrome $s_1 = wH$.*
2. *Si $wt(s_1) \leq 3$ entonces $u = [s_1, \mathbf{0}]$.*
3. *Si $wt(s_1 + b_i) \leq 2$ para alguna fila b_i de B entonces $u = [s_1 + b_i, e_i]$.*

4. *Calculamos el segundo síndrome* $s_2 = s_1 B$.
5. *Si* $wt(s_2) \leq 3$ *entonces* $u = [\boldsymbol{0}, s_2]$.
6. *Si* $wt(s_2 + b_i) \leq 2$ *para alguna fila* b_i *de* B *entonces* $u = [e_i, s_2 + b_i]$.
7. *Si* u *no ha sido determinado entonces se pide retransmisión.*

En el caso que se determine u *entonces la palabra enviada es* $v = w + u$.

Cerremos está sección viendo un ejemplo donde implementamos este algoritmo:

Ejemplo 3.5.3. *Decodificar* $w = 101111101111, 010010010010$.

Solución
Calculamos el síndrome para w, así tenemos:

$$\begin{aligned} s_1 = wH &= 101111101111 + 001111101110 \\ &= 100000000001. \end{aligned}$$

Como vemos $wt(s_1) = 2 \leq 3$ entonces podemos encontrar el patrón de error u:

$$u = [s_1, \mathbf{0}] = 100000000001, 000000000000.$$

Y por lo tanto podemos concluir que la palabra que fue enviada es $v = w + u = 001111101110, 010010010010$. Debido a que la matriz G está en forma estándar es fácil deducir que los primeros 12 dígitos son el mensaje enviado, así el mensaje que se envió es 001111101110.

En la última sección estudiaremos el código de Golay y como estos se relacionan con el código C_{24}.

3.6. Un código perfecto más - Código de Golay

Recordemos el ejemplo 3.2.4 este ejemplo demostraba la existencia de un código perfecto con parámetros $(23, k, 7)$. De hecho si al código c_{24} en su matriz generadora G la última columna la eliminamos generaremos un nuevo código con un matriz generado de la forma:

$$G' = [I_{12}, B'].$$

Donde B' es obtenida de la matriz B de tal manera que la última columna de B es eliminada. De la forma en que hemos creado la matriz G' podemos deducir la siguiente información $n = 23$ y $k = 12$. Otra cosa importante es que al eliminarle la última columna las palabras que forman las filas de G' tendrán peso impar, esto quiere decir que por el teorema 3.4.4 el nuevo código tiene distancia $d = 8 - 1 = 7$ y de hecho el código que genera

G' es nada más ni nada menos que el código que el ejemplo 3.2.4 aseguró que existiría **el código de Golay**. Por lo tanto el código de Golay que es generado por G' con parámetros $(23, 12, 7)$ es denotado por C_{23} (y es por esto que la sección donde se presento C_{24} se titulaba código extendido de Golay, ya que C_{24} es el código extendido de C_{23}).

Al ser C_{23} un código perfecto este corrige todos los patrones de peso menor o igual que 3 y ningún otro más. Podemos usar el algoritmo 3.5.2 para crear un algoritmo para C_{23}, para esto si tenemos una palabra w como está es una palabra de tamaño 23, entonces para usar C_{24} solo le agregamos un dígito a la palabra w de tal manera que el peso de w sea impar y luego implementar el algoritmo 3.5.2 y al resultado obtenido le quitamos el ultimo dígito y así sabremos cuál fue la palabra enviada. Así tenemos

Algoritmo 3.6.1. ***(IMLD para*** C_{23}***)*** *Sea w una palabra recibida. Entonces para calcular el patrón de error de w seguimos los siguientes pasos:*

1. *Formamos la palabra $w0$ o $w1$ de tal manera que tenga peso impar.*
2. *Decodificamos wi (ya sea $i = 0$ o $i = 1$) usando el algoritmo 3.5.2 para obtener una palabra de código v de C_{24}.*
3. *Removemos el ultimo dígito de v.*

Observaciones: Debido que hacemos que w al final tenga peso impar este nunca será una palabra del código C_{24} ya que en este solo hay palabras de peso par. Una ultima curiosidad de esta forma de operar es que si la palabra w es de C_{23} entonces cuando le agregamos el dígito de tal manera que el peso se impar entonces C_{24} detectará que solo hay problema en el último dígito ya que ese es el que monitorea la paridad de las palabras.

Finalmente hemos llegado a descubrir un tipo de código que se comporta bien y que nos permite corregir como máximo tres errores, también hemos aprendido como solo con un dígito mejorar las cualidades del código, aunque quizás todo este esfuerzo se vea pequeño en el sentido de solo poder corregir tres errores es un gran paso ya que a partir de aquí podemos ir mejorando nuestros códigos a modo que puedan corregir más errores. En resumen el capítulo 2 presenta lo básico para descubrir más tipos de códigos y sus ventajas sobre los códigos perfectos. Además si el lector se pregunta sobre una aplicación de estos códigos pues podemos decir que en los años 80 el código C_{24} fue utilizado en las misiones de la Voyager permitiendo la transmisión de fotografías de Jupiter y Saturno.

Apéndice A

Implementación en Octave de C_{24}

En el capítulo 3 se presentó el algoritmo 3.5.2 para el código C_{24} que permite corregir un máximo de 3 errores a la vez. Se ha programado dicho algoritmo en el software Octave para mostrar la eficiencia del algoritmo.

Para poder implementar el algoritmo primero tenemos que crear ciertas funciones. La primera de estas funciones extras es la función *wt* (función peso y recibe como parámetro una palabra y devuelve su peso) esta función calcula el peso de una palabra de tamaño n, la función está definida de la siguiente manera:

```
function s=wt(v)

s=0;

for i=1:length(v)

    s=s+v(i);

endfor
```

La segunda función es la que permite encontrar la fila de B que está en la posición donde hay un error, dicha función está nombrada como *buscar* (recibe como parámetro el síndrome y la matriz B y devuelve la posición del error) y sus lineas de código son:

```
function j=buscar(s,B)

j=0;
for i=1:12
```

```
    if wt(rem(s+B(i,:),2))<=2
      j=i;
      break
    endif
  endfor
```

La tercera función es la que crea un vector con la posición donde sucede un error (este error está en función de la posición que *buscar* encuentra) y está nombrada *ei* (recibe la posición del error y el tamaño de las palabras y devuelve la palabra con ceros menos en la posición del error) y está definida como:

```
  function e=ei(i,n)
  e=zeros(1,n);
  for j=1:n
    if j==i
      e(i)=1;
    endif
  endfor
```

Por último la función *mensaje* es la que muestra en pantalla las posiciones donde ocurre el error (recibe como parámetro la palabra patrón de error), dicha función está definida como:

```
  function mensaje(u)
  fprintf('Los dígitos cambiados estaban en la posicion:')
  for i=1:24
    if u(i)!=0
```

```
        i
    endif
endfor
```

Finalmente usando todas las cuatro funciones anteriores podemos crear el programa para el algoritmo 3.5.2, las lineas de código son:

```
function v=Golay(w)
w
u=[];
v=[];
i=0;
e=[];
B=[[1,1,0,1,1,1,0,0,0,1,0,1];
    [1,0,1,1,1,0,0,0,1,0,1,1];
    [0,1,1,1,0,0,0,1,0,1,1,1];
    [1,1,1,0,0,0,1,0,1,1,0,1];
    [1,1,0,0,0,1,0,1,1,0,1,1];
    [1,0,0,0,1,0,1,1,0,1,1,1];
    [0,0,0,1,0,1,1,0,1,1,1,1];
    [0,0,1,0,1,1,0,1,1,1,0,1];
    [0,1,0,1,1,0,1,1,1,0,0,1];
    [1,0,1,1,0,1,1,1,0,0,0,1];
```

```
    [0,1,1,0,1,1,1,0,0,0,1,1];
    [1,1,1,1,1,1,1,1,1,1,1,0]];
I=[[1,0,0,0,0,0,0,0,0,0,0,0];
    [0,1,0,0,0,0,0,0,0,0,0,0];
    [0,0,1,0,0,0,0,0,0,0,0,0];
    [0,0,0,1,0,0,0,0,0,0,0,0];
    [0,0,0,0,1,0,0,0,0,0,0,0];
    [0,0,0,0,0,1,0,0,0,0,0,0];
    [0,0,0,0,0,0,1,0,0,0,0,0];
    [0,0,0,0,0,0,0,1,0,0,0,0];
    [0,0,0,0,0,0,0,0,1,0,0,0];
    [0,0,0,0,0,0,0,0,0,1,0,0];
    [0,0,0,0,0,0,0,0,0,0,1,0];
    [0,0,0,0,0,0,0,0,0,0,0,1]];
H=[I;B];
s1=rem(w*H,2);
i=buscar(s1,B);
e=ei(i,12);
if wt(s1)<=3
    u=[s1,0,0,0,0,0,0,0,0,0,0,0,0];
```

```
elseif i!=0
    if wt(rem(s1+B(i,:),2))<=2
        u=[rem(s1+B(i,:),2),e];
    endif
endif
s2=rem(s1*B,2);
i=buscar(s2,B);
e=ei(i,12);
if wt(s2)<=3
    u=[0,0,0,0,0,0,0,0,0,0,0,0,s2];
elseif i! =0
    if wt(rem(s2+B(i,:),2))<=2
        u=[e,rem(s2+B(i,:),2)];
    endif
endif
if length(u)==0
    fprintf('Se necesita retransmision.')
else
    v=rem(w+u,2);
    mensaje(u)
endif
```

El programa Golay recibe como parámetro la palabra recibida y devolverá la palabra corregida. Procedamos a ver un resultado en específico donde usamos Golay, tomemos la palabra de código: [1,0,1,1,1,0,1,1,1,0,1,1,0,1,1,1,1,0,1,1,0,0,1,0] y cambiemos el primero y último dígito y así tendremos la palabra: [0,0,1,1,1,0,1,1,1,0,1,1,0,1,1,1,1,0,1,1,0,0,1,1] y al aplicar Golay tendremos el siguiente resultado en pantalla:

```
Golay([0,0,1,1,1,0,1,1,1,0,1,1,0,1,1,1,1,0,1,1,0,0,1,1])

w =

   0 0 1 1 1 0 1 1 1 0 1 1 0 1 1 1 1 0 1 1 0 0 1 1
Los digitos cambiados estaban en la posicion:

i = 1

i = 24

ans =

   1 0 1 1 1 0 1 1 1 0 1 1 0 1 1 1 1 0 1 1 0 0 1 0
```

Y como lo habíamos mencionado antes el problema estaban en el primero y último dígito (posición 1 y 24 respectivamente) y el programa cambia los valores en estas posiciones y así tendremos la palabra correcta del código, enviado originalmente. El programa es funcional y todos los cambios que hagamos hasta un máximo de 3 errores a la vez en una palabra de código serán corregidos correctamente por el programa Golay.

Conclusiones

1. *Dado un conjunto de palabras (que se usa para comunicar información) es capaz el código de detectar y corregir errores.*

2. *Construida la matriz de control de paridad somos capaces de saber en que posiciones los dígitos fueron cambiados por el ruido y por tanto poder corregir estos errores.*

3. *Usando los límites (tanto cota superior como inferior para la cardinalidad del código) somos capaces de saber si un código es posible que exista o no, suponiendo la dimensión del código lineal (si necesitamos un código lineal), el tamaño de las palabras y la distancia que deseamos que tenga.*

4. *Los códigos en forma estándar proporcionan una mayor manipulación para trabajar con ellos; esto se debe a que los algoritmos para encontrar la matriz de control de paridad así como poder decodificar son sumamente fáciles, de hecho basta con una simple inspección.*

5. *Al permutar dígitos de las palabras de un código creamos un código equivalente al anterior, pero esta permutación no afecta las propiedades de detección y corrección de errores, por ello a partir de un código no estándar podemos obtener uno que sí lo sea y que además tenga las propiedades de detección y corrección de errores de nuestro código original. Por lo tanto siempre podremos trabajar con un código estándar, cuando sea necesario.*

6. *Los códigos perfectos tienen la características que todas las palabras que no sean del código pueden ser detectadas y corregidas esto se debe a que al cumplir la igualdad del límite de Hamming hacemos que cada palabra del K^n esté en uno y solo un conjunto de distancia a lo sumo t de una palabra de código; en otras palabras lo que hemos logrado es crear un cubierta de bolas de centro una palabra de código y radio t, donde t es el entero que cumple que la distancia del código es $2t + 1$.*

7. *Desgraciadamente no existen muchos códigos perfectos; ya que los únicos que existen corrigen un máximo de un error o un máximo de tres errores a la vez (resultado que no se logro demostrar), por ello que dan las puertas abiertas para hacer más estudios en teoría de código para encontrar y crear códigos que permitan una mayor capacidad de corrección.*

Bibliografía

[1] D. R. Hankerson, Gary Hoffman, Douglas A. Leonard, Charles C. Lindner, Kevin T. Phelps, Chris A. Rodger, and James R. Wall. *Coding Theory and Cryptography: The Essentials*. CRC Press, 2000.

[2] Steven Roman. *Introduction to Coding and Information Theory*. Springer Verlag, 1996.

[3] Steven Roman. *Coding and Information Theory*, volume 134. Springer Science & Business Media, 1992.

[4] Vera Pless. *Introduction to the Theory of Error-Correcting Codes*, volume 48. John Wiley & Sons, 2011.