UNIVERSIDAD MICHOACANA DE SAN NICOLÁS DE HIDALGO

FACULTAD DE CIENCIAS FÍSICO - MATEMÁTICAS "Luis Manuel Rivera Gutiérrez"

TEORÍA Y SIMULACIÓN DE CÓDIGOS QUE CORRIGEN ERRORES

PARA OBTENER EL TÍTULO DE:

LICENCIADO EN CS. FÍSICO-MATEMÁTICAS

PRESENTA:

Omar Díaz Orozco

ASESOR:

Dr. Humberto Cárdenas Trigos

MORELIA, MICHOACÁN, NOVIEMBRE DE 2011

Agradecimientos

En primer lugar le doy gracias a toda mi familia por su apoyo en especial a mi mamá Ana Lila Orozco Valencia y a mi papá Juan David Díaz Villagómez gracias a ellos soy la persona que soy hoy en día y todos mis logros hasta la fecha son en gran parte debido a ellos dos.

A mi asesor y amigo el Dr. Humberto Cárdenas Trigos ya que sin él este trabajo no lo hubiera podido realizar, muchas gracias por toda la paciencia y el apoyo que me brindó, he aprendido muchas cosas de usted y no sólo de matemáticas sino de la vida en general ya que usted es de los mejores matemáticos y también una de las mejores personas que he conocido.

También quiero agradecer a las personas que revisaron mi trabajo y ayudaron a mejorarlo ellos son: Dr. Mario César Suárez Arriaga, Dr. Luis Valero Elizondo, Dr. José Gerardo Tinoco Ruíz, y Dr. José Carlos Cortés Zavala.

Por último en general gracias a toda la Facultad en Cs. Físico Matemáticas Mat. "Luis Manuel Rivera" de la Universidad Michoacana de San Nicolás de Hidalgo.

Contenido

Introducción **3**

1. Propiedades Generales de los Códigos **9**

2. Códigos Lineales **12**

2.1. Códigos de Hamming 15

2.2 Códigos de Hamming con Mathematica 17

3. Códigos Reed-Muller **21**

3.1. Monomios y vectores sobre $\mathbb{F}_2$ 21

3.2. Una visión simple de los códigos Reed-Muller 23

3.3. Código de Hadamard con Mathematica 27

4. Códigos Reed-Solomon **31**

4.1. Perspectiva original 31

4.2. Perspectiva clásica (códigos Reed-Solomon como códigos BCH) 34

4.3. Equivalencia de las dos formulaciones 35

4.4. El algoritmo de Euclides. 36

4.5. Corrección de errores en los códigos Reed-Solomon 40

4.6. Demostración de la corrección de errores en los códigos Reed-Solomon. .44

4.7. Códigos Reed-Solomon binarios 48

4.8. Códigos Reed-Solomon con GAP 50

5. Códigos Convolutivos **62**

5.1. Estados de un código 64

5.2. Codificación de una secuencia de entrada 65

5.3. Diagrama de estado 69

5.4. Diagrama de Trellis 71

5.5. Decodificación de Viterbi 72

5.6. Códigos convolutivos con Mathematica 78

6. Conclusiones **82**

RESUMEN

A grandes rasgos, codificar es transformar una información en una señal convenida para su comunicación. Decodificar sería el proceso inverso y complementario del anterior por el cual la señal comunicada es transformada en la información original. El auge de las comunicaciones a partir de la segunda mitad del siglo XX motivó un fuerte desarrollo de la teoría de códigos.

En teoría de la códigos, con aplicaciones en ciencias de la computación y las telecomunicaciones, la detección y corrección de errores son las técnicas que permiten la entrega confiable de datos digitales a través de canales de comunicación. Muchos canales de comunicación están sujetos a ruido del canal, y por lo tanto errores pueden ser introducidos durante la transmisión de la fuente a un receptor. Técnicas de detección de errores permiten detectar este tipo de errores, mientras que los códigos que corrigen errores permiten la reconstrucción de los datos originales siempre y cuando el número de errores sea menor de los que puede corregir el código.

En este trabajo se muestra como se pueden construir algunos de los primeros códigos utilizados en la práctica por la *NASA* (*National Aeronautics and Space Administration*) de los Estados Unidos para transmitir fotografías desde el espacio exterior, se muestra la teoría matemática que hay detrás de estos códigos y como es que funcionan, también incorpora la simulación de transmisiones de imágenes utilizando algunos códigos para mostrar la gran diferencia que estos hacen cuando la información se manda sobre canales ruidosos sin codificar.

Introducción

La teoría de códigos es una de las aplicaciones más recientes del álgebra. Se suele determinar su nacimiento en el año 1948 con los trabajos de Claude Shannon sobre la Teoría Matemática de la Información. Rachard Hamming, uno de los creadores de la teoría de códigos, cuenta la siguiente anécdota: Cuando trabajaba para Bell Laboratories, en los años cuarenta, tenía acceso a las computadoras sólo los fines de semana. Acostumbraba dejar corriendo ciertos programas en el ordenador y cuando volvía, el fin de semana siguiente, encontraba que algunos de los programas que más necesitaba no habían sido ejecutados (cuando el ordenador detecta un error en un programa, suele detener su ejecución y pasa a otro que esté en la cola de espera). Esto le ocasionaba importantes retrasos en su trabajo lo que le llevo a plantearse el problema de acondicionar de algún modo la información que maneja el ordenador de tal manera que no sólo fuera capaz de detectar un error y pararse, sino que pudiera corregir los errores.

Supongamos que se desea enviar información digital a través de un canal de comunicación de una forma rápida y segura. El canal de comunicación puede ser una línea telefónica, comunicación vía satélite, etc. A menudo ocurre que el mensaje recibido no concuerda con el enviado debido a errores humanos, interferencias, deficiencias en el equipo, etc. Se suele decir que la comunicación se hace a través de un canal con ruido.

Lo códigos que corrigen errores pretenden que el error, en caso de que ocurra pueda ser corregido por el receptor. Esto se consigue añadiendo cierta información redundante de forma que el mensaje original pueda ser recuperado a pesar de que haya ocurrido algún error. Estamos acostumbrados con el principio de redundancia del lenguaje cotidiano. Las palabras de nuestro lenguaje forman una pequeña parte de todas las cadenas posibles de letras. Por consiguiente, un error de imprenta en una palabra larga se reconoce porque la palabra se transforma en algo que se asemeja más a la palabra correcta de lo que se asemeja cualquier otra palabra que conocemos.

Uno de los objetivos de la Teoría de Códigos es conseguir aumentar todo lo que sea posible la probabilidad de que el mensaje correcto pueda ser recuperado a pesar de los posibles errores, siempre que el número de éstos sea razonable.

La Teoría de Códigos se usa en la transmisión de fotografías desde el espacio profundo. Cuando la señal emitida desde la nave espacial llega a Tierra es muy débil, porque ha recorrido una enorme distancia, y hay que amplificarla. Esto puede ocasionar que algún bit sea interpretado erróneamente, dando como resultado una fotografía de baja calidad. Para ver cómo la codificación consigue la mejora de la calidad de esas fotografías ver las imágenes 1 y 2.

Es interesante hacer una breve historia de cómo se ha desarrollado el proceso:

El *Mariner 4* (1965) envió a la Tierra 22 fotografías de Marte. En cada una de ellas se consideraba un reticulado de 200 $\times$ 200 cuadrados. A cada uno de ellos se le asigna una cadena de 6 ceros y unos que representa el grado de negro (000000 = blanco, 111111 = negro y una gama de 62 tonalidades de grises). Los números naturales desde el 0 hasta el 63 escritos en forma binaria se representan con cadenas de 6 ceros y unos. Precisamente, la representación binaria de 63 es 111111. El número total de bits por fotografía era de $6 \times 200 \times 200 = 240.000$. Llevó 8 horas transmitir cada fotografía.

Mariner 9 (1971) envió fotografías de Marte mucho mejores. Hay tres razones importantes para dicha mejora:

a) Ahora cada fotografía se descompone en 700×832 elementos.

b) *Mariner 9* fue el primero en orbitar en torno a Marte.

c) Se implemento el código de Hadamard. Las cadenas de 6 ceros y unos que representan el grado de gris de cada elemento de la fotografía se codificó como una cadena de 32 (26 bits redundantes). Este código es capaz de corregir hasta 7 dígitos erróneos en cada cadena.

Aunque el código de Hadamard fue el primero desarrollado para aplicación espacial, no fue en realidad el primero que se puso en marcha. Un sistema de codificación basado en un código convolutivo (2,1,20) con polinomios generadores

$$g_1(D) = 1$$

$$g_2(D) = 1 + D + D^2 + D^5 + D^6 + D^8 + D^9 + D^{12} + D^{13}$$

fue lanzado a bordo del *Pioneer 9* en noviembre 8 de 1968 dirigido hacia la órbita solar.

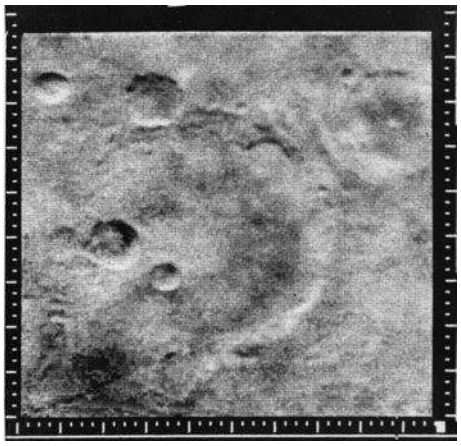


Imagen 1. Cráteres marcianos vistos desde la *Mariner 4*.

La *Viking 1* (1976) implemento nuevamente el código de Hadamard, esta sonda espacial se poso sobre Marte y envió fotografías a color de gran calidad (el método fue similar al del caso de fotografías en blanco y negro).

El uso de la decodificación de probabilidad máxima de Viterbi (*maximum-likelihood Viterbi decoding*) dio origen a la siguiente generación de sistemas de comunicaciones del espacio profundo.

Las misiones espaciales *Voyager 1* y *2* se pusieron en marcha en 1977 para explorar Júpiter y Saturno. Ambos utilizan un código convolutivo no sistemático (2,1,7) con polinomios generadores

$$g_1(D) = 1 + D + D^3 + D^4 + D^6$$

$$g_2(D) = 1 + D^3 + D^4 + D^5 + D^6$$

este código y un código (3,1,7) con polinomios generadores

$$g_1(D) = 1 + D + D^3 + D^4 + D^6$$

$$g_2(D) = 1 + D^3 + D^4 + D^5 + D^6$$

$$g_3(D) = 1 + D^2 + D^4 + D^5 + D^6$$

fueron ambos aprobados como los Códigos Planetarios Estándar de la NASA/ESA por el *Consultative Committee on Space Data Systems* (CCSDS). El código (2,1,7) también se ha empleado en muchas otras aplicaciones, incluyendo la comunicación por satélite y la telefonía celular. Los códigos anteriores fueron decodificados usando un decodificador de Viterbi. En la sonda *Voyager 2*, además del código convolutivo, se apoya a la sonda con la implementación de un código Reed-Solomon: la concatenación Reed-Solomon-

Viterbi (RSV) permitió una corrección de errores muy poderosa, y ha permitido ampliar el viaje de la nave hacia Urano y Neptuno.

La CCSDS actualmente recomienda el uso de códigos que corrigen errores con un rendimiento similar al Código RSV implementado en la *Voyager 2* como mínimo. Los códigos concatenados son cada vez menos populares en las misiones espaciales, y son reemplazados por códigos más poderosos, como los códigos Turbo o códigos LDPC (*Low-density parity-check code*).

Los diferentes tipos de misiones espaciales y misiones orbitales que se llevan acabo sugieren que tratar de encontrar un sistema de corrección de errores con una talla para todos será un problema constante por algún tiempo. Para misiones cerca de la tierra la naturaleza del ruido del canal es diferente al de una nave espacial en una misión interplanetaria. Además, a medida que una nave espacial aumenta su distancia de la Tierra, el problema de la corrección de ruido se hace más grande.

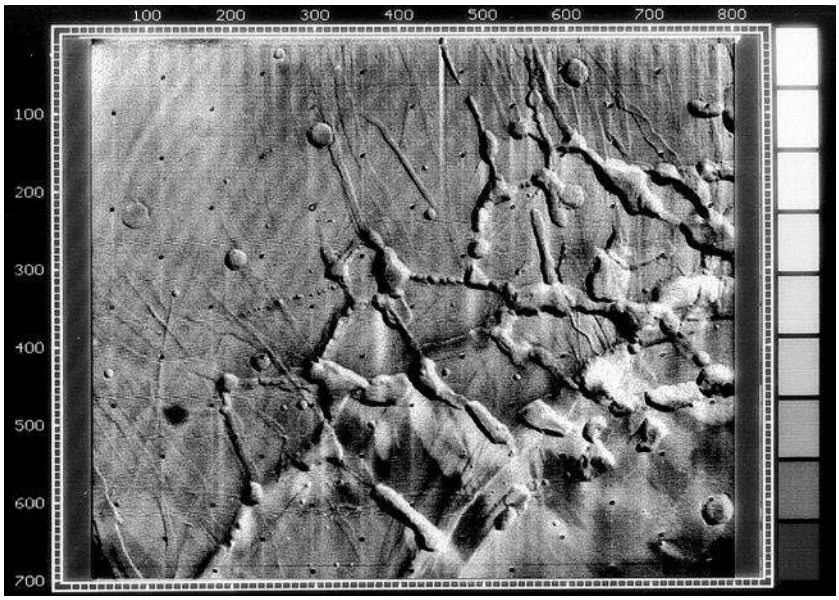


Imagen 2. Los Noctis Labyrinthus vistos desde la Mariner 9.

Otras aplicaciones importantes de los códigos que corrigen errores son:

- La Internet: en una típica pila TCP / IP, control de errores se realiza a varios niveles:

  - Cada estructura Ethernet lleva una suma de comprobación CRC-32. Estructuras recibidas con comprobación de suma incorrecta se descartan por el hardware del receptor.

- El encabezado de IPv4 contiene una suma de comprobación para proteger el contenido de la cabecera. Los paquetes con sumas de comprobación incorrecta son abandonados dentro de la red o en el receptor.

- La suma de comprobación no se implementa en la cabecera IPv6 con el fin de minimizar los costos de procesamiento en la red de rutas y porque la tecnología actual de la capa de enlace se supone que proporciona la detección de errores suficiente.

- El UDP (*User Datagram Protocol*) tiene una suma de comprobación opcional cubriendo efectos nocivos y dirigiendo la información de la UDP y cabeceras IP. Los paquetes con sumas de comprobación incorrecta se descartan por la pila de la red del sistema operativo. La suma de comprobación es opcional con IPv4, sólo, porque la suma de comprobación de la capa IP ya puede proporcionar el nivel deseado de protección de errores.

- El TCP (*Transmission Control Protocol*) proporciona una suma de comprobación para la protección de cargas dañinas y dirigiendo la información de la TCP y las cabeceras IP. Los paquetes con sumas de comprobación incorrecta se descartan en la pila de red, y eventualmente es retransmitida mediante un ARQ (*Automatic Repeat reQuest*).

- La transmisión vía satélite: la demanda de ancho de banda de transpondedor de satélite continúa creciendo, impulsado por el deseo de ofrecer la televisión y datos IP. La disponibilidad y limitaciones del ancho de banda del transpondedor han limitado este crecimiento, porque la capacidad de transpondedor está determinada por el esquema de modulación seleccionado y la tasa de corrección de errores. Los códigos de Reed Solomon y Viterbi se han utilizado durante casi 20 años para la entrega de televisión digital por satélite.

- Almacenamiento de datos: los códigos que corrigen errores mejoran la fiabilidad de los medios de almacenamiento de datos. Los códigos Reed Salomón se utilizan en discos compactos para corregir los errores causados por rayones. Los discos duros modernos utilizan códigos CRC para detectar errores y códigos Reed-Solomon para corregir errores menores en el sector de lecturas, y para recuperar datos de los sectores que se han dañado y almacenar esos datos en los sectores de repuesto.

- Corrección de errores de memoria: la memoria DRAM puede proporcionar una mayor protección contra los errores de software basándose en códigos que corrigen errores. Tal memoria de corrección de errores, conocida como memoria EDAC protegida, es particularmente deseable para aplicaciones de alta tolerancia a fallos, tales como servidores. Controladores de corrección de errores de memoria, tradicionalmente, utilizan códigos de Hamming.

1. Propiedades Generales de los Códigos

Asumiendo que la información es codificada utilizando un alfabeto Q con q símbolos distintos. Un código es llamado *código de bloque* si la información codificada puede ser dividida en bloques de n símbolos que pueden ser decodificados independientemente. Estos bloques son las *palabras código* y n es llamado la longitud de bloque o longitud de palabra.

Definición. Si $\mathbf{x} \in \mathrm{Q}^n$, $\mathbf{y} \in \mathrm{Q}^n$, entonces la *distancia* d(x, y) de $\mathbf{x}$ y $\mathbf{y}$ esta definida por:

$$d(x, y) := |\{i | 1 \leq i \leq n, x_i \neq y_i\}|$$

El *peso* $w(\mathbf{x})$ de $\mathbf{x}$ es definido como:

$$w(\mathbf{x}) := d(\mathbf{x}, \mathbf{0})$$

en donde (0, 0, ... , 0) es denotado como $\mathbf{0}$.

La distancia anteriormente definida es llamada la *distancia de Hamming,* que es de hecho una métrica en Q^n. Si se utiliza un canal con la propiedad de que un error en la posición i no influye en otras posiciones y un símbolo de error puede estar en cada uno de los restantes q - 1 símbolos con igual probabilidad, entonces la distancia de Hamming es una buena manera de medir el error contenido de un mensaje recibido.

Un código de C es un subconjunto propio no vacío de Q^n. Si $|C| = 1$ lo llamamos código trivial, si $q = 2$ el código se llama un código binario, para $q = 3$ un código ternario, etc. Los siguientes conceptos desempeñan un papel esencial en la teoría de códigos.

Definición. La *distancia mínima* de un código no trivial C es

$$min\{d(\mathbf{x}, \mathbf{y}) | \mathbf{x} \in C, \mathbf{y} \in C, \mathbf{x} \neq \mathbf{y}\}.$$

El *peso mínimo* de C es

$$min\{w(\mathbf{x}) | \mathbf{x} \in C, \mathbf{x} \neq \mathbf{0}\}.$$

Determinar el número de errores que están garantizados de ser únicamente corregidos en un código dado es una parte importante de la teoría de códigos. Para ello, en general, considere lo siguiente. Para $\mathbf{x} \in \mathbb{Q}^n$ y un entero positivo r, definimos

$$S_r(\mathbf{x}) = \{\mathbf{y} \in Q^n | d(\mathbf{x}, \mathbf{y}) \leq r\}.$$

En la terminología estándar $S_r(\mathbf{x})$ es llamada la bola de radio r alrededor de $\mathbf{x}$. Sea C un código con distancia mínima d, y sea t el mayor entero tal que $t < \mathrm{d}/2$. Entonces $S_t(\mathbf{x}) \cap S_t(\mathbf{x})$ es vacío para cada par de $\mathbf{x}$, $\mathbf{y}$ distintos elementos de C. Si $\mathbf{z}$ es un vector recibido en $\mathbb{Q}^n$ con $d(\mathbf{u},\mathbf{z}) \leq \mathrm{t}$ para algún $\mathbf{u} \in \mathrm{C}$, entonces $\mathbf{z} \in S_t(\mathbf{u})$, y $\mathbf{z} \notin S_t(\mathbf{v})$ para todos los demás $\mathbf{v} \in \mathrm{C}$. Es decir, si un vector recibido $\mathbf{z} \in \mathbb{Q}^n$ difiere de un elemento del código $\mathbf{u} \in \mathrm{C}$ en t o menos posiciones, entonces cualquier otro elemento del código C diferirá de $\mathbf{z}$ en más de t posiciones. Por lo tanto, el cálculo del vecino más cercano siempre permite que sean corregidos t o menos errores en el código. Se dice que C es un código que corrige t errores.

Suponiendo que C es un código en $\mathbb{Q}^n$ que corrige t errores. Ahora nos dirigimos al problema de determinar el número de vectores en V que son garantizados de ser corregidos en C. Nótese en primer lugar que para cualquier $\mathbf{x} \in \mathbb{Q}^n$, hay $\binom{n}{i} \cdot (q-1)^i$ vectores en $\mathbb{Q}^n$ que difieren de $\mathbf{x}$ en exactamente i posiciones. Además cualquier vector en $\mathbb{Q}^n$ que difiere de $\mathbf{x}$ en i posiciones estará en $S_t(\mathbf{x})$ para $i \leq t$. Por lo tanto, el número de vectores en $S_t(\mathbf{x})$ será:

$$\binom{n}{0} + \binom{n}{1} \cdot (q-1) + \cdots + \binom{n}{t} \cdot (q-1)^t.$$

Para determinar el número de vectores en V que están garantizados de ser corregibles en C, se deben contar únicamente el número de vectores que están en $S_t(\mathbf{x})$ a medida que $\mathbf{x}$ varía sobre los elementos del código C. Dado que los conjuntos $S_t(\mathbf{x})$ son disjuntos dos a dos, el número de vectores en V que difieren de un elemento del código C en t o menos posiciones, y están consecuentemente garantizados de ser únicamente corregibles en C, es:

$$|C| \cdot \sum_{i=0}^{t} \binom{n}{i} \cdot (q-1)^i.$$

El hecho de que $|C| = q^n$ da paso al siguiente teorema, que da una cota en el número de vectores en $\mathbb{Q}^n$ que están garantizados de ser corregibles en un código que corrige t errores. Esta cota se llama la *cota de Hamming.*

Teorema. Supongamos que C es un código que corrige t errores en $\mathbb{Q}^n$. Entonces

$$|C| \cdot \sum_{i=0}^{t} \binom{n}{i} \cdot (q-1)^i \leq q^n.$$

Un código C en $\mathbb{Q}^n$ se dice que es *perfecto* si cada vector en $\mathbb{Q}^n$ esta garantizado de ser corregible en C. Es decir, un código C en $\mathbb{Q}^n$ es perfecto si en la desigualdad del teorema anterior se cumple la igualdad.

Teorema. Podemos establecer la siguiente cota para el número máximo de palabras $A_q(n,d)$ de un código de largo n y distancia mínima d:

$$A_q(n,d) \leq q^{n-d+1}$$

esta cota se llama la *cota de Singleton.*

Demostración.

Consideremos las palabras obtenidas al borrar las primeras $d-1$ coordenadas de cada palabra código. Como la distancia entre dos palabras del código es al menos d, las palabras obtenidas son todas distintas y es claro que no hay más que q^{n-d+1} de ellas pues tienen largo $n-d+1$.

Definición. Diremos que un código es MDS (maximum distance separable) si alcanza la cota de Singleton.

En la práctica es común desear construir códigos que tengan un largo número de palabras código y que estén garantizados para corregir un gran número de errores. Sin embargo, el número de errores garantizados de ser corregibles en un código está obviamente relacionado con el número de palabras código. De hecho, la construcción de un código C en $\mathbb{Q}^n$ que corrija t errores para valores fijos de n y t tal que se maximice |C| ha sido un problema importante de reciente interés matemático.

2. Códigos Lineales

Pasando ahora al problema de construir códigos que tengan alguna estructura algebraica. La primera idea idea es tomar un grupo $\mathbb{Q}$ como alfabeto y tomar un subgrupo C de $\mathbb{Q}^n$ como código. Esto se llama un *código de grupo.* Sin embargo en esta sección necesitaremos más estructura. En lo siguiente $\mathbb{Q}$ es un campo finito con q elementos denotado $\mathbb{F}_q$, donde $q = p^r$ (p primo). Entonces $\mathbb{Q}^n$ es un espacio vectorial de dimensión n que a menudo se denota como $\mathscr{R}^{(n)}$.

Definición. Un *código lineal* C es un subespacio vectorial de $\mathscr{R}^{(n)}$. Si C tiene dimensión k entonces C es llamado un código $[n, k]$.

De ahora en adelante se utilizará la notación código $[n, k, d]$ para un código lineal k −dimensional de longitud n y distancia mínima d.

Definición. Una *matriz generadora* G de un código lineal C es una matriz con k filas y n columnas en la que las filas son una base de C.

Si C es una matriz generadora de C entonces $C = \{\mathbf{a}G \mid \mathbf{a} \in \mathbb{Q}^k\}$. Diremos que G esta en su *forma estándar* (a menudo llamada forma escalón reducida) si $G = (I_k \quad P)$, donde I_k es la matriz identidad de $k \times k$.

La capacidad de corregir errores de dos códigos C_1 y C_2 es igualmente buena si C_2 es obtenido aplicando una permutación fija de los símbolos a todas las palabras del código de C_1. A tales códigos los llamamos *equivalentes.* Algunas veces la definición de equivalencia es extendida dando la permutación de los símbolos de $\mathbb{Q}$. Es bien sabido del álgebra lineal que todo código lineal es equivalente a un código con matriz generadora en forma escalón reducida.

Si un código C tiene distancia mínima $d = 2e + 1$, entonces corrige hasta e errores en una palabra recibida. Si $d = 2e$ un patrón de error de peso e es siempre detectado. En general si C tiene M palabras se deben verificar $\binom{M}{2}$ pares de palabras del código para encontrar d. Para códigos lineales el trabajo es más simple.

Teorema. Para un código lineal C la distancia mínima es igual al peso mínimo.

Demostración.

$d(\boldsymbol{x}, \boldsymbol{y}) = d(\boldsymbol{x} - \boldsymbol{y}, \mathbf{0}) = w(\boldsymbol{x} - \boldsymbol{y})$ y si $\mathbf{x} \in C$, $\mathbf{y} \in C$ entonces $\mathbf{x} - \mathbf{y} \in C$.

∎

Definición. Si C es un código $[n, k]$ el *código dual* $C^{\perp}$ es definido como

$$C^{\perp} := \{\mathbf{y} \in \mathscr{R}^{(n)} | \ \forall_{\mathbf{x} \in C}[\langle \mathbf{x}, \mathbf{y} \rangle = 0]\}.$$

El código dual $C^{\perp}$ es un código lineal, a saber, un código $[n, n - k]$. Si $C = C^{\perp}$ entonces C es llamado un código *auto-dual*.

Si $G = (I_k \ \ P)$ es una matriz generadora en su forma estándar de un código C entonces $H = (-P^{\top} \ \ I_{n-k})$ es una matriz generadora de $C^{\perp}$. Esto se deriva del hecho de que H tiene el tamaño y rango correcto y que $GH^{\top} = 0$ implica que toda palabra $\mathbf{a}G$ tiene producto interno igual a 0 con cada fila de H. En otras palabras tenemos

$$\mathbf{x} \in C \Leftrightarrow \mathbf{x}H^{\top} = \mathbf{0}.$$

Si $\mathbf{y} \in C^{\perp}$ entonces la ecuación $\langle \mathbf{x}, \mathbf{y} \rangle = 0$ que es valida para todos los $\mathbf{x} \in C$ es llamada ecuación de *verificación de paridad*. H es llamada *matriz de control de paridad* de C.

Definición. Si C es un código lineal con matriz de verificación de paridad H entonces para todo $\mathbf{x} \in \mathscr{R}^{(n)}$ llamamos a $\mathbf{x}H^{\top}$ el *síndrome* de $\mathbf{x}$.

Se a visto anteriormente que las palabras del código están caracterizadas por tener síndrome igual a **0.** El síndrome es una ayuda importante para decodificar vectores recibidos. Ya que C es un subgrupo de $\mathscr{R}^{(n)}$ podemos hacer una partición $\mathscr{R}^{(n)}$ en clases laterales de C. Dos vectores $\mathbf{x}$ y $\mathbf{y}$ están en están en la misma clase lateral si y sólo si tienen el mismo síndrome ($\mathbf{x}H^{\top} = \mathbf{y}H^{\top} \Leftrightarrow \mathrm{x} - \mathrm{y} \in C$). Por lo tanto si un vector $\mathbf{x}$ es recibido con un patrón de errores $\mathbf{e}$ entonces $\mathbf{x}$ y $\mathbf{e}$ tienen el mismo síndrome. Se sigue que para la decodificación de probabilidad máxima de $\mathbf{x}$ se debe escoger un vector $\mathbf{e}$ de

peso mínimo en la clase lateral que contiene a $\mathbf{x}$ y luego decodificar $\mathbf{x}$ como $\mathbf{x} - \mathbf{e}$. El vector $\mathbf{e}$ es llamado *líder de la clase lateral.*

Ejemplo. Sea C un código binario [4, 2] de matriz generadora

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

entonces $C = \{(0,0,0,0), (1,0,1,1), (0,1,0,1), (1,1,1,0)\}$ y las clases laterales de C son

$$\begin{aligned}
(0,0,0,0) + C &= \{(0,0,0,0), (1,0,1,1), (0,1,0,1), (1,1,1,0)\} \\
(1,0,0,0) + C &= \{(1,0,0,0), (0,0,1,1), (1,1,0,1), (0,1,1,0)\} \\
(0,1,0,0) + C &= \{(0,1,0,0), (1,1,1,1), (0,0,0,1), (1,0,1,0)\} \\
(0,0,1,0) + C &= \{(0,0,1,0), (1,0,0,1), (0,1,1,1), (1,1,0,0)\}.
\end{aligned}$$

Podemos observar que este código tiene distancia mínima 2 y que, usando el algoritmo descrito, permite corregir un error si este es cometido en una de las primeras 3 posiciones (no en la cuarta).

Aquí vemos la primer gran ventaja de introducir la estructura algebraica. Para un código $[n, k]$ sobre $\mathbb{F}_q$ hay q^k palabras del código y q^n posibles mensajes recibidos. Asumiendo que el tiempo necesario para la transmisión de la información es alto. El receptor necesita saber los q^{n-k} líderes de las clases laterales correspondientes a todos los posibles síndromes. Ahora q^{n-k} es mucho más chico que q^n. Si el código no tuviera estructura entonces para cada posible palabra recibida $\mathbf{x}$ se tendría que hacer una lista de la palabra transmitida más probable.

Está claro que si C tiene distancia mínima $d = 2e + 1$ entonces todo patrón de errores de peso $\leq e$ es el único líder de la clase lateral de alguna clase lateral porque dos vectores con peso $\leq e$ tienen distancia $\leq 2e$ y por lo tanto están en diferentes clases laterales. Si C es perfecto entonces no hay otros líderes de las clases laterales. Si un código C tiene distancia mínima $2e + 1$ entonces todos los líderes de las clases laterales tienen peso $\leq e + 1$ entonces el código es llamado *cuasi-perfecto.*

A menudo resultará útil añadir un símbolo extra a cada palabra de algún código C de acuerdo a una regla natural. La más común de estas se da en la siguiente definición.

Definición. Si C es un código de longitud n sobre $\mathbb{F}_q$ definimos el *código extendido* $\overline{C}$ como

$$\overline{C} := \left\{ (c_1,\ c_2, \ldots,\ c_n,\ c_{n+1}) | (c_1, \ldots,\ c_n) \in C,\ \sum_{i=1}^{n+1} c_i = 0 \right\}.$$

Si C es un código lineal con matriz generadora G y matriz de control de paridad H entonces $\overline{C}$ tiene matriz generadora $\overline{G}$ y matriz de control de paridad $\overline{H}$, donde $\overline{G}$ es obtenida añadiendo una columna a G de tal manera que la suma de las columnas de $\overline{G}$ es 0 y donde

$$\overline{H} := \begin{pmatrix} 1 & 1 & 1 & \ldots & 1 \\ & & & & 0 \\ & & & & 0 \\ & & H & & 0 \\ & & & & \vdots \\ & & & & 0 \end{pmatrix}.$$

Si C es un código binario con distancia mínima impar d entonces $\overline{C}$ tiene distancia mínima $d + 1$ ya que todos los pesos y distancias para $\overline{C}$ son pares.

2.1. Códigos de Hamming

Sea G una matriz generadora de $k \times n$ de un código $C\ [n,\ k]$ sobre $\mathbb{F}_q$. Si cualquiera dos columnas de G son linealmente independientes, entonces C es llamado un *código proyectivo*. El código dual $C^{\perp}$ tiene a G como matriz de control de paridad. Si $\mathbf{c} \in C^{\perp}$ y si $\mathbf{e}$ es un error de peso 1, entonces el síndrome $(\mathbf{c} + \mathbf{e})G^{\top}$ es un múltiplo de una columna de G. Dado que esto determina de forma única la columna de G resulta que $C^{\perp}$ es un código que corrige por lo menos un error. Veamos ahora el caso en el que la n es máxima dado k.

Definición. Sea $n := (q^k - 1)/(q - 1)$. El *código de Hamming* $[n,\ n - k]$ sobre $\mathbb{F}_q$ es un código para el cual la matriz de control de paridad tiene cualquier par de columnas

linealmente independientes, es decir, las columnas son el máximo conjunto en el que cualquier pareja de vectores es linealmente independiente sobre $\mathbb{F}_q$.

Recordemos que un código lineal C tiene distancia mínima d entonces el peso mínimo también es d. Sea $\mathbf{c} \in$ C una palabra de peso mínimo y H la matriz de control de paridad de C. Como $\mathbf{c}H^\top = \mathbf{0}$ esto quiere decir que H tiene d columnas que son linealmente dependientes. Por otro lado si H tuviese un conjunto menor de columnas linealmente dependientes entonces la relación de dependencia lineal nos daría una palabra de peso menor a d en el código.

En particular, si buscamos códigos con distancia mínima 3 (el mínimo necesario para corregir un error), la matriz de control de paridad correspondiente tendrá columnas que satisfacen que cualquiera dos de ellas son linealmente independientes y que hay 3 de ellas que son linealmente dependientes. Claramente la distancia mínima de un código de Hamming es igual a 3.

Ejemplo. El código de Hamming binario [7, 4] tiene matriz de control de paridad

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

Si consideramos dos columnas de H y la suma de estas dos (por ejemplo las primeras dos columnas de H), entonces hay una palabra de peso 3 en C con unos en las posiciones correspondientes a estas columnas (por ejemplo (1, 1, 1, 0, 0, 0, 0)). Con una inspeción de H se puede ver que el código extendido $\overline{C}$ es auto-dual

Teorema. Los códigos de Hamming son códigos perfectos.

Demostración.

Sea C un código de Hamming $[n,\ n-k]$ sobre $\mathbb{F}_q$, donde $n := (q^k - 1)/(q-1)$. Si $\mathbf{x} \in C$ entonces

$$q^{n-k} \cdot \sum_{i=0}^{1} \binom{n}{i} \cdot (q-1)^i = q^{n-k}(1 + n(q-1)) = q^{n-k} \cdot q^k = q^n.$$

2.2. Códigos de Hamming con *Mathematica*

Construcción del código:

```
t4 = Tuples[{0, 1}, 4];

f[z_, v_, w_, t_] := {Mod[z + v + t, 2], Mod[z + w + t, 2], Mod[z,2],
Mod[v + w + t, 2], Mod[v, 2], Mod[w, 2], Mod[t, 2]};

For[i = 1, i < 17, i++, h[i] = Apply[f, t4[[i]]]]

H = Array[h, 16];
```

Progama que simula la transmisión de una imagen usando el código de Hamming:

Este programa simula la transmisión de una imagen codificada con el código de Hamming y otra sin codificar con probabilidad de error en cada posición de 1/20, de esta manera se pueden comparar los resultados.

```
HamTrans = Function[data = ImageData[#1, Byte];

   t1 = Dimensions[data][[1]]; t2 = Dimensions[data][[2]];
   s = data[[1]]; For[j1 = 1, j1 < t1, j1++, s = Union[s,
Union[data[[j1]], data[[j1 + 1]]]]];

   f1 = Function[

     For[i = 0, i < 17, i++, If[#1 === s[[i]], h1 = H[[i]];
Break[]]]; h1];

   inf1 =
    Function[
     For[i4 = 1, i4 < 17, i4++,
      If[#1 === H[[i4]], h2 = s[[i4]]; Break[], h2 = s[16]]]; h2];
```

```
   tran = Function[
     For[i1 = 1, i1 < t2 + 1, i1++, y[i1] = f1[#1[[i1]]]];

     Y = Array[y, t2];

     k = 0; m = 0;
     Do[
      a = 20;

      For[i = 0, i < 8, i++, x = Random[Integer, {1, a}];
          If[x == 1, β[i] = 1, β[i] = 0]];

      o = Array[β, 7];

      Suma =
       Mod[o + p,
        2];   (*Vector con posibles errores*)

(*Evaluar vector en el sistema de ecuaciones*)

      Ne = Mod[({
           {1, 0, 1, 0, 1, 0, 1},
           {0, 1, 1, 0, 0, 1, 1},
           {0, 0, 0, 1, 1, 1, 1}
          }).Suma, 2];

      (*Detectar la posicion donde esta el error*)

      Posicion = Ne.{1, 2, 4};
      ε = Suma;
      k = k + 1;

      If[Posicion > 0, Suma[[Posicion]] = Mod[Suma[[Posicion]] + 1,
2]];
      Ta[k] = Suma;

      If[k == t2, k = 0], {p, Y}];

      A = Array[Ta, t1];

     For[i3 = 1, i3 < t2 + 1, i3++, v[i3] = inf1[A[[i3]]]];

     V = Array[v, t2]];

   f2 = Function[
     For[i = 0, i < 17, i++, If[#1 === s[[i]], h3 = t4[[i]];
Break[]]]; h3];
```

```
   inf2 =
    Function[
     For[i4 = 1, i4 < 17, i4++, If[#1 === t4[[i4]], h2 = s[[i4]];
Break[]]];h2];

   tran2 = Function[
     For[i0 = 1, i0 < t2 + 1, i0++, y[i0] = f2[#1[[i0]]]];

     Y = Array[y, t2];

     k = 0; m = 0;

     Do[
      a = 20;

      For[i = 0, i < 5, i++, x = Random[Integer, {1, a}];
          If[x == 1, β[i] = 1, β[i] = 0]];

      o = Array[β, 4];

      Suma = Mod[o + p, 2];
      k = k + 1;
      Ta[k] = Suma;
      If[k == t2, k = 0], {p, Y}];

     T = Array[Ta, t2];

     For[i3 = 1, i3 < t2 + 1, i3++, v[i3] = inf2[T[[i3]]]];

     V = Array[v, t2]];

   l = {}; l1 = {}; For[u = 1, u < t1 + 1, u++, l = Append[l,
tran[data[[u]]]]; l1 = Append[l1, tran2[data[[u]]]]; im = Image[l,
Byte]; im1 = Image[l1, Byte]]];
```

Para utilizar este programa introducimos una imagen con contiene 16 tonalidades de gris y escribimos por ejemplo:

```
{Dynamic[im],Dynamic[im1]}
```

```
HamTrans[
```

```
]
```

y obtenemos las siguientes imágenes. La primera imagen que fue codificada con el código de Hamming contiene 1260 errores y la segunda que no fue codificada contiene 5041 errores y así es como se verían


La diferencia no es tan notable pero recodemos que este código solo corrige un error.

3. Códigos Reed-Muller

Los códigos Reed-Muller se encuentran entre los más antiguos y más conocidos códigos que corrigen errores. Fueron descubiertos y propuesto por D. E. Muller y I. S. Reed en 1954. Como se menciono anteriormente el código de Hadamard que es un caso especial de los códigos Reed-Muller fue utilizado en la *Mariner 9* para transmitir imágenes en blanco y negro de Marte a la Tierra. Los códigos de Reed-Muller son relativamente fáciles de decodificar, y los códigos de primer orden son especialmente eficientes.

3.1 Monomios y vectores sobre $\mathbb{F}_2$

Definición. Si $\mathbf{a} = (a_0,\ a_1,...,\ a_n)$ y $\mathbf{b} = (b_0,\ b_1,...,\ b_n)$ son vectores de $\mathbb{F}_2^n$ la *multiplicación de vectores* esta dada por

$$\mathbf{a} * \mathbf{b} := (a_0b_0, a_1b_1, \dots a_nb_n).$$

Notación. En esta sección se utiliza una cadena de longitud n con elementos de $\mathbb{F}_2$ para escribir un vector en el espacio vectorial $\mathbb{F}_2^n$ por brevedad cuando esto es inequívoco. Por ejemplo el vector (1, 0, 1, 1, 0, 1, 0) simplemente se escribirá como 1011010.

Considerando el anillo $\mathscr{R}_m = \mathbb{F}_2[x_1,\ x_2,...,\ x_m]$. Se mostrará mas adelante que existe una biyección entre los elementos de $\mathscr{R}_m$ y $\mathbb{F}_2^{2^m}$.

Un *polinomio booleano* es una combinación lineal de *monomios booleanos* con coeficientes en $\mathbb{F}_2$. Un monomio booleano p en variables $x_1,...,\ x_n$ es una expresión de la forma

$$\mathbf{p} = x_1^{r_1}x_2^{r_2} \dots\ x_m^{r_m}\ \textit{donde}\ r_i \in \{0, 1, 2, \dots\}\ y\ 1 \leq i \leq m.$$

Cualquier elemento de $\mathscr{R}_m$ puede ser considerado como un polinomio booleano.

La *forma reducida* $\mathbf{p}'$ de $\mathbf{p}$ se obtiene aplicando las reglas,

$$x_i x_j = x_j x_i \ y \ x_i^2 = x_i,$$

hasta que los factores sean distintos. Un polinomio booleano esta en su forma reducida si cada monomio esta en su forma reducida. El *grado* de un polinomio booleano $\mathbf{q}$ es el grado ordinario de su forma reducida $\mathbf{q}'$.

Un ejemplo de un polinomio booleano en su forma reducida de grado tres es

$$\mathbf{q}' = x_1 + x_2 + x_1 x_2 + x_2 x_3 x_4.$$

Considere el siguiente mapeo $\psi : \mathscr{R}_m \to \mathbb{F}_2^{2^m}$ definido como sigue:

$$\psi(0) = \underbrace{00\ldots0}_{2^m}$$

$$\psi(1) = \underbrace{11\ldots1}_{2^m}$$

$$\psi(x_1) = \underbrace{11\ldots1}_{2^{m-1}}\underbrace{00\ldots0}_{2^{m-1}}$$

$$\psi(x_2) = \underbrace{11\ldots1}_{2^{m-2}}\underbrace{00\ldots0}_{2^{m-2}}\underbrace{11\ldots1}_{2^{m-2}}\underbrace{00\ldots0}_{2^{m-2}}$$

$$\psi(x_3) = \underbrace{11\ldots1}_{2^{m-3}}\underbrace{00\ldots0}_{2^{m-3}}\underbrace{11\ldots1}_{2^{m-3}}\underbrace{00\ldots0}_{2^{m-3}}\underbrace{11\ldots1}_{2^{m-3}}\underbrace{00\ldots0}_{2^{m-3}}\underbrace{11\ldots1}_{2^{m-3}}\underbrace{00\ldots0}_{2^{m-3}}$$

$$\vdots \qquad\qquad \vdots$$

$$\psi(x_i) = \underbrace{11\ldots1}_{2^{m-i}}\underbrace{00\ldots0}_{2^{m-i}}\ldots$$

$$\vdots \qquad\qquad \vdots$$

Para calcular $\psi(\mathbf{p})$, para cualquier monomio $\mathbf{p} \in \mathscr{R}_m$ encontramos su forma reducida:

$$\mathbf{p}' = x_{i_1} x_{i_2} \dots x_{i_r}$$

$$(\textit{donde } i_j \in \mathbb{Z}_m,\ i_j = i_k \Rightarrow j = k,\ \textit{y } 0 \leq r \leq m).$$

Entonces:

$$\psi(\mathbf{p}) = \psi(x_{i_1}) * \psi(x_{i_2}) * \cdots * \psi(x_{i_r})$$

Para cualquier polinomio $\mathbf{q} \in \mathscr{R}_m$ podemos escribir $\mathbf{q}$ como:

$$\mathbf{q} = m_1 + m_2 + \cdots + m_r$$

$$(\textit{donde } m_i \textit{ es un monomio booleano},\ m_i = m_j \Rightarrow i = j,\ \textit{y } 0 \leq r \leq 2^m).$$

Entonces:

$$\psi(\mathbf{q}) = \psi(m_1) + \psi(m_2) + \cdots + \psi(m_r).$$

Ejemplo. Sea $\mathbf{p} = 1 + x_2 + x_1^5 x_3^2 + x_1 x_2^4 x_3^{101} \in \mathscr{R}_3$, y como $\mathbf{p}$ tiene forma reducida $\mathbf{p}' = 1 + x_2 + x_1 x_3 + x_1 x_2 x_3$ entonces:

$$\begin{aligned}
\psi(\mathbf{p}) &= \psi(\mathbf{p}') \\
&= \psi(1) + \psi(x_2) + \psi(x_1 x_3) + \psi(x_1 x_2 x_3) \\
&= \psi(1) + \psi(x_2) + \psi(x_1) * \psi(x_3) + \psi(x_1) * \psi(x_2) * \psi(x_3) \\
&= 11111111 + 11001100 + 11110000 * 10101010 + 11110000 * 11001100 * 10101010 \\
&= 00010011
\end{aligned}$$

Proposición. ψ es una biyección, y de hecho, ψ es un homomorfismo de anillos. Por lo tanto, $\mathscr{R}_m$ y $\mathbb{F}_2^{2^n}$ son isomorfos, y a partir de este momento nos referiremos a los vectores y sus polinomios asociados en forma reducida indistintamente.

3.2. Una visión simple de los códigos Reed-Muller

Definición. El código de Reed-Muller de orden r, denotado $\mathscr{R}(r,\ m)$, es el conjunto de todos los polinomios de grado a lo más r en el anillo $\mathscr{R}_m$. Alternativamente, a través del isomorfismo ψ puede ser pensado como un subespacio de $\mathbb{F}_2^{2^m}$.

El código de Reed-Muller de orden $0\,\mathscr{R}(0,\,m)$ consiste de los monomios $\{0,\,1\}$, que es equivalente a los siguientes vectores:

$$\{\underbrace{00\ldots0}_{2^m},\underbrace{11\ldots1}_{2^m}\}.$$

Por otra parte, El código de Reed-Muller de orden $0\,\mathscr{R}(m,\,m)=\mathscr{R}_n\cong\mathbb{F}_2^{2^m}$.

Para el código de Reed-Muller $\mathscr{R}(r,\,m)$, definimos la matriz generadora de la siguiente manera:

$$G_{\mathscr{R}(r,m)}=\begin{pmatrix}\psi(1)\\ \psi(x_1)\\ \psi(x_2)\\ \vdots\\ \psi(x_m)\\ \psi(x_1x_2)\\ \psi(x_1x_3)\\ \vdots\\ \psi(x_{m-1}x_m)\\ \psi(x_1x_2x_3)\\ \vdots\\ \psi(x_{m-r+1}x_{m-r+2}\ldots x_m)\end{pmatrix}.$$

Ejemplo. La matriz generadora para un código $\mathscr{R}(1,\,3)$ es:

$$G_{\mathscr{R}(1,3)}=\begin{pmatrix}\psi(1)\\ \psi(x_1)\\ \psi(x_2)\\ \psi(x_3)\end{pmatrix}=\begin{pmatrix}1&1&1&1&1&1&1&1\\ 1&1&1&1&0&0&0&0\\ 1&1&0&0&1&1&0&0\\ 1&0&1&0&1&0&1&0\end{pmatrix}.$$

Ejemplo. La matriz generadora para un código $\mathscr{R}(2,\,3)$ es:

$$G_{\mathscr{R}(2,3)} = \begin{pmatrix} \psi(1) \\ \psi(x_1) \\ \psi(x_2) \\ \psi(x_3) \\ \psi(x_1x_2) \\ \psi(x_1x_3) \\ \psi(x_2x_3) \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix}.$$

Ejemplo. La matriz generadora para un código $\mathscr{R}(3, 3)$ es:

$$G_{\mathscr{R}(3,3)} = \begin{pmatrix} \psi(1) \\ \psi(x_1) \\ \psi(x_2) \\ \psi(x_3) \\ \psi(x_1x_2) \\ \psi(x_1x_3) \\ \psi(x_2x_3) \\ \psi(x_1x_2x_3) \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

Ejemplo. La matriz generadora para un código $\mathscr{R}(2, 4)$ es:

$$G_{\mathscr{R}(2,4)} = \begin{pmatrix} \psi(1) \\ \psi(x_1) \\ \psi(x_2) \\ \psi(x_3) \\ \psi(x_4) \\ \psi(x_1x_2) \\ \psi(x_1x_3) \\ \psi(x_1x_4) \\ \psi(x_2x_3) \\ \psi(x_2x_4) \\ \psi(x_3x_4) \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix}.$$

Teorema. La matriz $G_{\mathscr{R}(r,m)}$ tiene dimensión $k \times n$, donde:

$$k = \sum_{i=0}^{r} \binom{m}{i}$$

$$n = 2^m$$

por lo tanto la dimensión del código es k.

Demostración. Las filas de la matriz pueden dividirse por el grado del monomio que representan; para el código $\mathscr{R}(r,\ m)$, hay $\binom{m}{0}$ filas correspondientes a los monomios de grado 0, $\binom{m}{1}$ filas correspondientes a los monomios de grado 1, $\binom{m}{2}$ filas correspondientes a monomios de grado 2, etc...

Por lo tanto, hay:

$$\binom{m}{0} + \binom{m}{1} + \cdots + \binom{m}{r} = \sum_{i=0}^{r} \binom{m}{i}$$

filas, y por lo tanto, la dimensión del código es k. Por la definición de ψ es fácil ver que el número de columnas es 2^m.

Teorema. El dual de $\mathscr{R}(r,\ m)$ es $\mathscr{R}(m - r - 1,\ m)$.

Demostración.
a) Por el teorema anterior la dimensión de $\mathscr{R}(r,\ m)$ es

$$\binom{m}{0} + \binom{m}{1} + \cdots + \binom{m}{r}$$

y la dimensión de $\mathscr{R}(m - r - 1,\ m)$ es

$$\binom{m}{0} + \binom{m}{1} + \cdots + \binom{m}{m-r-1}$$

y se sabe que

$$\binom{m}{k} = \binom{m}{m-k}$$

entonces

$$\mathscr{R}(r,m) + \mathscr{R}(m-r-1,m) = \binom{m}{0} + \binom{m}{1} + \cdots + \binom{m}{m} = 2^m = n.$$

b) Sea u unos de los renglones de la matriz generadora de $\mathscr{R}(r, m)$ y sea v unos de los renglones de la matriz generadora de $\mathscr{R}(m - r - 1, m)$, por definición de ψ el producto $u * v$ tiene peso par, es decir, los vectores u, v son ortogonales.

Corolario. El código $\mathscr{R}(m - 2, m)$ es el código de Hamming extendido $[n, n - m - 1]$.

Los códigos Reed-Muller pueden ser decodificados con la decodificación lógica de la mayoría (*majority logic decoding*) pero en nuestro caso utilizaremos la decodificación de de probabilidad máxima (*maximum likelihood decoding*).

3.3. Código de Hadamard con *Mathematica*

Construcción del código:

```
V = Tuples[{0, 1}, 5];

Ω = Union[Range[31], {0}];

f = Function[V[[#1]]];

b = Function[β = {};
   For[i = 1, i < 33, i++, If[f[i][[#1]] == 1, β = Union[β, {i -
1}]]]; β];

iB[1] = Ω;

For[j = 2, j < 7, j++, iB[j] = b[j - 1]];

B = Array[iB, 6];

ad = Function[Complement[Union[#1, #2], Intersection[#1, #2]]];

lc = Function[L = {};
   For[i = 1, i < Length[#1] + 1, i++, Do[L = ad[L, B[[n]]], {n,
{#1[[i]]}}]];
    L];
```

```
M = Subsets[Range[6]];

For[k = 1, k < 65, k++, iH[k] = lc[M[[k]]]];

(*Codigo de Hadamard*)

H = Array[iH, 64];
```

Progama que simula la transmisión de una imagen usando el código de Hadamard:

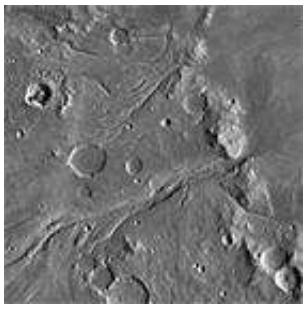
```
HadTrans = Function[

   data = ImageData[#1, Byte];

   t1 = Dimensions[data][[1]]; t2 = Dimensions[data][[2]];

   s = data[[1]];
   For[j1 = 1, j1 < t1, j1++, s = Union[s, Union[data[[j1]], data[[j1
+ 1]]]]];

   f1 = Function[
     For[i = 1, i < 65, i++, If[#1 === s[[i]], h = M[[i]]; Break[]]];
h];

   inf1 =
    Function[For[i = 1, i < 65, i++, If[#1 === M[[i]], h = s[[i]];
Break[]]];
     h];

   rand =
    Function[z = {};
     For[i = 0, i < 32, i++,
      If[Random[Integer, {1, #1}] == 1, z = Union[z, {i}]]]; z];

   rand2 =
    Function[z = {};
     For[i1 = 1, i1 < 7, i1++,
      If[Random[Integer, {1, #1}] == 1, z = Union[z, {i1}]]]; z];

   dcod =
      Function[l = 1;
       While[Length[ad[#1, H[[l]]]] > 7, l++; If[l > 64, l = 1;
Break[]]];M[[l]]];
```

```
tran = Function[

 For[i1 = 1, i1 < t1 + 1, i1++, y[i1] = f1[#1[[i1]]]];

  Y = Array[y, t1];

 k = 0;

 Do[
   k = k + 1;

   V = rand[20];

   Suma = ad[V, lc[p]];

   Ta[k] = dcod[Suma]; If[k == t2, k = 0], {p, Y}];

   T = Array[Ta, t1];

   For[i1 = 1, i1 < t1 + 1, i1++, r[i1] = inf1[T[[i1]]]];
  R = Array[r, t1]];

tran2 = Function[
 For[i0 = 1, i0 < t1 + 1, i0++, y[i0] = f1[#1[[i0]]]];

 Y = Array[y, t1];

 k = 0;

 Do[
   k = k + 1;

   V = rand2[20];

   Suma = ad[V, p]; Ta[k] = Suma, {p, Y}];

   T = Array[Ta, t1];

   For[i3 = 1, i3 < t1 + 1, i3++, r[i3] = inf1[T[[i3]]]];

   R = Array[r, t1]];

l = {}; l1 = {};

For[w = 1, w < t1 + 1, w++, l = Append[l, tran[data[[w]]]];
 l1 = Append[l1, tran2[data[[w]]]]; im = Image[l, Byte, ImageSize
-> 280]; im1 = Image[l1, Byte, ImageSize -> 280]]];
```

Para utilizar este programa introducimos una imagen con contiene 64 tonalidades de gris en este caso utilizaremos una de las imágenes enviadas por la *Mariner 9* y escribimos lo siguiente por ejemplo:

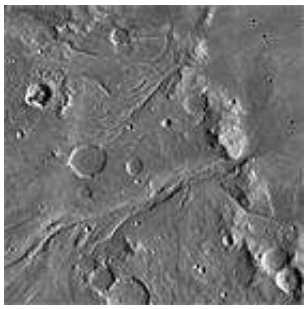
```
{Dynamic[im], Dynamic[im1]}
```

```
HadTrans[
```



```
]
```

y obtenemos las siguientes imágenes. La primera imagen que fue codificada con el código de Hadamard contiene 4 errores y la segunda que no fue codificada contiene 5249 errores y así es como se verían



La diferencia en este ejemplo es notable.

4. Códigos Reed-Solomon

Los códigos Reed-Solomon (códigos RS) se desarrollaron en 1960 por Irving S. Reed y Gustave Salomón, que eran entonces miembros del *MIT Lincoln Laboratory*. Su artículo seminal fue titulado "*Polynomial Codes over Certain Finite Fields*". Como se mencionó anteriormente en 1977, los códigos RS se implementan, especialmente, en el programa *Voyager* en forma de códigos concatenados. La primera aplicación comercial en productos de consumo fabricados en serie apareció en 1982 con el disco compacto, en el que dos códigos RS intercalados se utilizan. Hoy en día, los códigos de RS son ampliamente implementados en dispositivos de almacenamiento digital, tales como CDs, DVDs, discos Blu-ray, en las tecnologías de transmisión de datos, como DSL y WiMAX, en los sistemas de difusión como DVB y ATSC, y en aplicaciones informáticas, tales como sistemas RAID 6.

Los códigos RS pueden ser vistos como un caso especial de códigos BCH, sin embargo, los códigos Reed-Solomon son significativamente más populares que los otros códigos BCH debido a que son ideales para corregir ráfagas de errores. Se dice que un vector recibido contiene una ráfaga de errores si contiene varios errores muy juntos. Hay muchas situaciones en las cuales los errores de transmisión ocurren naturalmente en ráfagas por ejemplo en telecomunicaciones y en cintas magnéticas.

4.1. Perspectiva original

El concepto original de codificación Reed-Solomon describe la codificación del mensaje de k símbolos como coeficientes de un polinomio $p(x)$ de grado máximo $k-1$ sobre un campo finito de orden N, y evaluar el polinomio en los $n > k$ distintos puntos de entrada. El muestreo de un polinomio de grado $k-1$ en más de k puntos crea un sistema sobredeterminado, y esto permite al receptor la recuperación del polinomio dada cualquier k fuera de los n puntos de muestreo con la interpolación de Lagrange. La secuencia de puntos distintos se crea con un generador del grupo multiplicativo del campo finito, incluyendo el valor 0, lo que permite cualquier valor de n hasta N.

Es decir que si (x_1, x_2, ... , x_n) es la secuencia de entrada de n valores sobre el campo finito F, y definimos además el conjunto de polinomios de grado menor o igual a k con coeficientes en F como*:*

$$L_k = \{f \in F[x] \mid deg(f) \leq k\}$$

Entonces para cada $k \in \mathbb{Z}$ con $1 \leq k \leq n \leq N$, se define el código de Reed-Solomon:

$$C := \{(f(x_1), f(x_2), \ldots, f(x_n)) \mid f \in L_{k-1}\}.$$

Notemos que $C \subseteq F^n$ de manera que es un código sobre F. Además, la función ε: $L_{k-1} \to F^n$ dada por $\varepsilon(f) = (f(x_1), ..., f(x_n))$ es una transformación lineal cuya imagen es C de manera que se trata de un código lineal.

Como se describió anteriormente, una secuencia de entrada (x_1, x_2, ... , x_n) de $n = N$ valores se crea como (0, α^0, α^1, ... , α^{N-2}) donde α es una raíz primitiva de F. Cuando se omite el 0 de la secuencia y ya que $\alpha^{N-1} = 1$, se deduce que por cada polinomio $p(x)$ la función $p(\alpha x)$ es también un polinomio del mismo grado, y su palabra código es un cambio cíclico hacia la izquierda de la palabra código derivada de $p(x)$, por lo tanto los códigos Reed-Solomon pueden ser vistos como códigos cíclicos.

Propiedades generales

La dimensión de C es a lo más la dimensión de $L_{k-1} = k$.

Demostracion:

Veamos que el núcleo de ε es trivial. Si $\varepsilon(f) = 0$, entonces f tiene al menos n raíces, pero f tiene grado menor a $k \leq n$ de manera que $f = 0$. Esto demuestra que la dimensión de C es exactamente k.

∎

La distancia mínima de C es $n - k + 1$.

Demostración:

Supongamos que $f \neq 0$ y que $\varepsilon(f)$ tiene peso $d = d_{min}$. Entonces f tiene al menos $n - d$ ceros de manera que su grado debe ser también al menos $n - d$. Como $f \in L_{k-1}$ esto implica que $n - d \leq k - 1$, o equivalentemente, que $d \geq n - k + 1$. Pero ya sabemos por la cota de Singleton que $d \leq n - k + 1$ de manera que $d = n - k + 1$ y por lo tanto C es un código MDS.

∎

La capacidad de corrección de errores de un código RS es determinada por su distancia mínima, o equivalente, por $n - k$, la medida de la redundancia en el bloque. Un código RS puede corregir hasta $(n - k) / 2$ símbolos erróneos, es decir, el número de errores que corrige es la mitad de símbolos redundantes añadidos a el bloque.

Para usos prácticos de los códigos RS, es común el uso de un campo finito F con 2^m de elementos. En este caso, cada símbolo puede ser representado como un valor de m-bits. El remitente envía los puntos de datos como bloques codificados, y el número de símbolos en el bloque codificado es $n = 2^m - 1$.

Estas propiedades de los códigos RS los hacen especialmente adecuados para aplicaciones donde se producen errores en ráfagas. Esto se debe a que para el código no importa la cantidad de bits en error en un símbolo ya que si varios bits en un símbolo están dañados sólo cuenta como un solo error. Por el contrario, si una secuencia de datos no se caracteriza por ráfagas de errores, sino por los errores aleatorios de un solo bit, un código RS es generalmente una mala elección en comparación con un código binario.

4.2. Perspectiva clásica (códigos Reed-Solomon como códigos BCH)

En la práctica, en lugar de enviar los valores de la evaluación de un polinomio, en la codificación de símbolos se consideran los coeficientes de un polinomio de salida $s(x)$ construido al multiplicar el polinomio mensaje $b(x)$ de grado máximo $k - 1$ por un polinomio generador $g(x)$ de grado $t = N - k - 1$. El polinomio generador $g(x)$ se define por tener α, α^2, ..., α^t como sus raíces, es decir,

$$g(x) = (x - \alpha)(x - \alpha^2)... (x - \alpha^t) = g_0 + g_1 x + ... + g_{t-1} x^{t-1} + x^t .$$

El transmisor envía los $N - 1$ coeficientes de $s(x) = b(x)g(x)$, y el receptor puede usar la división de polinomios por $g(x)$ del polinomio recibido para determinar si el mensaje es un error, un residuo diferente de cero significa que un error fue detectado. Sea $r(x)$ el polinomio del residuo no cero, entonces el receptor puede evaluar $r(x)$ en las raíces de $g(x)$, y construir un sistema de ecuaciones que elimina $s(x)$ e identifica qué coeficientes de $r(x)$ están en un error, y la magnitud de cada coeficiente de error. Si el sistema de ecuaciones se puede resolver, el receptor sabe cómo modificar su $r(x)$ para obtener el más probable $s(x)$.

Ejemplo. Eligiendo el polinomio primitivo $p(x) = x^4 + x + 1 \in \mathrm{Z}_2[x]$. Los elementos no cero del campo $F = \mathrm{Z}_2[x]/(p(x))$ están listados en la siguiente tabla. Usando este campo F, obtenemos el siguiente polinomio generador para un código RS.

Potencia	Elemento del campo	Potencia	Elemento del campo
α^1	α	α^9	$\alpha^3 + \alpha$
α^2	α^2	α^{10}	$\alpha^2 + \alpha + 1$
α^3	α^3	α^{11}	$\alpha^3 + \alpha^2 + \alpha$
α^4	$\alpha + 1$	α^{12}	$\alpha^3 + \alpha^2 + \alpha + 1$
α^5	$\alpha^2 + \alpha$	α^{13}	$\alpha^3 + \alpha^2 + 1$

α^6	$\alpha^3 + \alpha^2$	α^{14}	$\alpha^3 + 1$
α^7	$\alpha^3 + \alpha + 1$	α^{15}	1
α^8	$\alpha^2 + 1$		

$$\begin{aligned} g(x) &= (x - \alpha)(x - \alpha^2)\ (x - \alpha^3)(x - \alpha^4) \\ &= ... \\ &= x^4 + \alpha^{13}x^3 + \alpha^6x^2 + \alpha^3x + \alpha^{10} \end{aligned}$$

Para construir una de las palabras código considere $b(x) = \alpha^{10}x^9 \in F[x]$.
Entonces

$$b(x)g(x) = \alpha^{10}x^{13} + \alpha^8x^{12} + \alpha x^{11} + \alpha^{13}x^{10} + \alpha^5x^9$$

es una de las palabras código.

4.3. Equivalencia de las dos formulaciones

A primera vista, estas dos definiciones de los códigos Reed-Solomon parecen muy diferentes. En la primera definición, las palabras código son los valores de los polinomios, mientras que en la segunda, son coeficientes. Por otra parte, los polinomios en la primera definición están obligados a ser de pequeño grado, mientras que en la segunda definición deben tener raíces específicas.

La equivalencia de las dos definiciones se prueba mediante la transformada de Fourier discreta. Esta transformación, que existe en todos los campos finitos, así como los números complejos, establece una dualidad entre los coeficientes de los polinomios y sus valores. Esta dualidad puede resumirse como sigue: Sea $p(x)$ y $q(x)$ dos polinomios de grado menor que n. Si los valores de $p(x)$ son los coeficientes de $q(x)$, entonces (hasta un factor escalar y reordenación), los valores de $q(x)$ son los coeficientes de $p(x)$.

Para que esto tenga sentido, los valores deben ser tomados en los lugares $x = \alpha^i$, para $i = 0, \ldots, n-1$, donde α es una raíz primitiva n-ésima de la unidad.

Para ser más precisos, sea

$$p(x) = v_0 + v_1 x + v_2 x^2 + \cdots + v_{n-1} x^{n-1}$$
$$q(x) = f_0 + f_1 x + f_2 x^2 + \cdots + f_{n-1} x^{n-1}$$

y asumiendo que $p(x)$ y $q(x)$ están relacionados por la transformada de Fourier discreta. Entonces, los coeficientes y los valores de $p(x)$ y $q(x)$ se relacionan de la siguiente manera: para todo $i = 0, \ldots, n-1$, $f_i = p(\alpha^i)$ y $v_i = 1/n \; q(\alpha^{n-i})$.

Usando estos hechos, tenemos: $(f_0, f_1, \ldots, f_{n-1})$ es una palabra código del código Reed-Solomon de acuerdo con la primera definición

- si y sólo si p(x) es de grado menor que k (porque $f_0, \ldots, f_{n-1}$ son los valores de $p(x)$),
- si y sólo si $v_i = 0$ para $i = k, \ldots, n-1$,
- si y sólo si $q(\alpha^i) = 0$ para $i = 1, \ldots,$ n $- k$ (porque $q(\alpha^i) = nv_{n-i}$),
- si y sólo si $(f_0, f_1, \ldots, f_{n-1})$ es una palabra del código Reed-Solomon de acuerdo con la segunda definición.

Esto demuestra que las dos definiciones son equivalentes.

4.4. El algoritmo de Euclides

Antes de proceder con la demostración de la corrección de errores en los códigos RS veamos el algoritmo de Euclides ya que es de vital importancia para el esquema de corrección de errores, es interesante como un algoritmo que tiene más de 20 siglos de antigüedad puede ser utilizado para decodificar códigos que corrigen errores que aun hoy en día se utilizan.

Sean a y b elementos distintos de cero en un dominio Euclidiano D, y considere un elemento $d \in$ D para el que $d \mid a$ y $d \mid b$. Supongamos que para todo $x \in D$, si $x \mid a$ y $x \mid b$, entonces $x \mid d$. Entonces d es llamado *máximo común divisor* de a y b. Usaremos la notación $d = (a, b)$ para representar esto.

Es bien sabido que si a y b son elementos distintos de cero en un dominio Euclidiano D. Entonces existe el máximo común divisor d de a y b y que se puede expresar como $d = au + bv$ por algún $u, v \in D$. Para calcular el máximo común divisor de dos enteros o polinomios se puede utilizar el algoritmo de Euclides. Describiremos este algoritmo a continuación.

Sean a y b elementos distintos de cero en un dominio euclidiano D, y sea N el conjunto de los enteros no negativos. Puesto que D es un dominio euclidiano, entonces hay un mapeo δ: $D^* \to N$ para el que podemos encontrar q_1, $r_1 \in D$ con $a = bq_1 + r_1$ y $r_1 = 0$ o $\delta(r_1) < \delta(b)$. Supongamos que $\delta(r_1) < \delta(b)$. Entonces podemos encontrar q_2, $r_2 \in D$ con $b = r_1q_2 + r_2$ y $r_2 = 0$ o $\delta(r_2) < \delta(r_1)$. Supongamos que $\delta(r_2) < \delta(r_2)$. Entonces podemos encontrar q_3, $r_3 \in D$ con $r_1 = r_2q_3 + r_3$ y $r_3 = 0$ o $\delta(r_3) < \delta(r_2)$. Continuamos este proceso hasta la primera vez $r_i = 0$ (que está garantizado a pasar eventualmente ya que $\delta(r_i)$ forma una sucesión estrictamente decreciente de números enteros no negativos). Es decir, construimos todos los q_i, r_i de las siguientes ecuaciones.

$$\begin{aligned}
a &= bq_1 + r_1 \quad \{\delta(r_1) < \delta(\mathrm{b})\} \\
\mathrm{b} &= r_1q_2 + r_2 \quad \{\delta(r_2) < \delta(r_1)\} \\
r_1 &= r_2q_3 + r_3 \quad \{\delta(r_3) < \delta(r_2)\} \\
&\vdots \\
r_{n-2} &= r_{n-1}q_n + r_n \quad \{\delta(r_n) < \delta(r_{n-1})\} \\
r_{n-1} &= r_nq_{n+1} + 0
\end{aligned}$$

Trabajando con esta lista de ecuaciones, podemos ver que r_n divide ambos a y b. También podemos ver que todo $x \in D$ que divide a ambos a y b también debe dividir r_n. Por lo tanto, $(a, b) = r_n$. Esta técnica para determinar (a, b) se llama el *algoritmo de Euclides.*

Hemos mostrado una técnica para la determinar del máximo común divisor (a, b) de dos números enteros o polinomios a y b. Todavía falta mostrar una técnica para encontrar u y v tales que $(a, b) = au + bv$. Para ello, se considera la siguiente tabla que se construye usando los q_i, r_i de la lista anterior de ecuaciones, y los u_i, v_i que

describimos a continuación. Vamos a llamar a este cuadro *tabla del algoritmo de Euclides.*

FILA	**Q**	**R**	**U**	**V**
-1	$-$	$r_{-1} = a$	$u_{-1} = 1$	$v_{-1} = 0$
0	$-$	$r_0 = b$	$u_0 = 0$	$v_0 = 0$
1	q_1	r_1	u_1	v_1
2	q_2	r_2	u_2	v_2
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
n	q_n	r_n	u_n	v_n

Las entradas en cada fila $i = 1, 2,..., n$ de esta tabla se construyen de la siguiente manera. Los q_i, r_i son de la ecuación i-ésima

$$r_{i-2} = r_{i-1}q_i + r_i \qquad (1.1)$$

en la lista anterior de ecuaciones. Note que si resolvemos para r_i en (1.1), obtenemos la siguiente ecuación.

$$r_i = r_{i-2} - r_{i-1}q_i \qquad (1.2)$$

A continuación se construyen u_i, v_i siguiendo este modelo para la construcción de r_i de q_i. En concreto, se construye u_i, v_i de q_i de la siguiente manera.

$$u_i = u_{i-2} - u_{i-1}q_i \qquad (1.3)$$
$$v_i = v_{i-2} - v_{i-1}q_i \qquad (1.4)$$

Muchas relaciones útiles existen entre las entradas de una tabla del algoritmo de Euclides. Por ejemplo, la siguiente ecuación se aplica a todas las filas i.

$$r_i = au_i + bv_i \qquad (1.5)$$

Claramente, esta ecuación se aplica a las filas $i = -1$ y 0. Para ver que es cierto para todas las filas siguientes, suponemos que es verdad para todas las filas a hasta la fila $k - 1$. Luego, utilizando (1.2), (1,3) y (1.4), se deduce que

$$\begin{aligned} r_k &= r_{k-2} - r_{k-1}\, q_k \\ &= (au_{k-2} + bv_{k-2}) - (au_{k-1} + bv_{k-1})q_k \\ &= a(u_{k-2} - u_{k-1}q_k) + b(v_{k-2} - v_{k-1}q_k) \\ &= au_k + bv_k. \end{aligned}$$

En concreto, $r_n = au_n + bv_n$. Pero recordemos, hemos dicho que $r_n = (a,\ b)$. Por lo tanto, para $u = u_n$ y $v = v_n$, tenemos $(a,\ b) = au + bv$.

Otra relación útil entre las entradas de la tabla del algoritmo de Euclides es la siguiente ecuación para todo $i = -1,\ 0,\ 1,\ 2,\ldots,\ n - 1$.

$$r_i u_{i+1} - u_i r_{i+1} = (-1)^i b \qquad (1.6)$$

Note en primer lugar que esta ecuación es claramente cierta para $i = -1$. Para ver que es verdad para todas las siguientes, supongamos que es cierto para $i = k - 1$. Entonces, utilizando (1.2), (1,3), y el hecho de que sumar un múltiplo de una fila de una matriz a otra fila en la matriz no cambia el determinante de la matriz, se deduce que

$$\begin{aligned} r_k u_{k+1} - u_k r_{k+1} &= \begin{vmatrix} r_k & u_k \\ r_{k+1} & u_{k+1} \end{vmatrix} \\ &= \begin{vmatrix} r_k & u_k \\ r_{k-1} - r_k q_{k+1} & u_{k-1} - u_k q_{k+1} \end{vmatrix} \\ &= \begin{vmatrix} r_k & u_k \\ r_{k-1} & u_{k-1} \end{vmatrix} \\ &= r_k u_{k-1} - u_k r_{k-1} \\ &= -(r_{k-1} u_k - u_{k-1} r_k) \\ &= -(-1)^{k-1} b \\ &= (-1)^k b. \end{aligned}$$

Dos relaciones adicionales que existen entre las entradas de una tabla del algoritmo de Euclides son las siguientes ecuaciones para todo $i = -1,\ 0,\ 1,\ 2,\ldots,\ n - 1$.

$$r_i v_{i+1} - v_i r_{i+1} = (-1)^{i+1} a \qquad (1.7)$$
$$u_i v_{i+1} - u_{i+1} v_i = (-1)^{i+1} \qquad (1.8)$$

Estas ecuaciones se pueden verificar de una manera similar a la verificación de (1.6) dada anteriormente.

4.5. Corrección de errores en los códigos Reed-Solomon

Sea F un campo de orden 2^n, y sea C un código RS. Supongamos que c(x) $\in$ C se transmite y se recibe el polinomio $r(x) \neq c(x)$ en $F[x]$ de grado menor que $2^n - 1$. Entonces $r(x) = c(x) + e(x)$ para algún polinomio error $e(x)$ distinto de cero en $F[x]$ de grado menor que $2^n - 1$. Para corregir $r(x)$ sólo se debe determinar $e(x)$, para lo cual podríamos calcular $c(x) = r(x) + e(x)$. Pero teniendo en cuenta que $c(\alpha^i) = 0$ para $i = 1, ..., n - k$ esto implica que $r(\alpha^i) = e(\alpha^i)$ para $i = 1, ..., n - k$. Por lo tanto, conociendo $r(x)$, también sabemos algo de información acerca de $e(x)$. Volveremos a llamar a los valores de $r(\alpha^i)$ los síndromes de $r(x)$.

En esta sección vamos a resumir e ilustrar un esquema de corrección de errores de los códigos Reed-Solomon. Usaremos la notación de $RS(2^n - 1, t)$ para representar un código Reed-Solomon que corrige t errores con palabras código de longitud $2^n - 1$.

Sea F un campo de orden 2^n, y sea C un código $RS(2^n - 1, t)$ en $F[x]$. Supongamos que c(x) $\in$ C es transmitido y se recibe el polinomio $r(x) = c(x) + e(x)$ para algún polinomio error $e(x)$ distinto de cero en $F[x]$ de grado menor que $2^n - 1$. Podemos utilizar los pasos siguientes para determinar $e(x)$.

1. En primer lugar, calcular los primeros $2t$ síndromes de $r(x)$, que denotaremos como $S_1 = r(\alpha)$, $S_2 = r(\alpha^2)$,..., $S_{2t} = r(\alpha^{2t})$, y formamos el siguiente polinomio síndrome $S(z)$.

$$S(z) = S_1 + S_2 z + S_3 z^2 + \cdots + S_{2t} z^{2t-1}$$

2. A continuación, se construye la tabla de algoritmo de Euclides para los polinomios $a(z) = z^{2t}$ y $b(z) = S(z)$ en $F[x]$ deteniéndose en la primera fila j para la que $deg(r_j) < t$.

3. A continuación se pueden encontrar las posiciones de error en $r(x)$ encontrando las raíces de $V(z)$. Específicamente, si α^{i_1}, α^{i_2}, ..., α^{i_k}. son raíces de $V(z)$, entonces $r(x)$ contiene errores en las las posiciones x^{-i_1}, x^{-i_2}, ..., x^{-i_k}. Por último, debemos encontrar los coeficientes de $e(x)$ en estas posiciones de error. Sea e_{-i} el coeficiente del termino x^{-i} en $e(x)$. Entonces:

$$e_{-i} = \frac{R(\alpha^i)}{V'(\alpha^i)}.$$

Ejemplo 1. Considere el polinomio primitivo $p(x) = x^4 + x^3 + 1$ en $Z_2[x]$. Para el elemento $\alpha = x$ en el campo $F = Z_2[x]/(p(x))$ de orden 16, una lista de los elementos del campo correspondientes a las primeras 15 potencias de α se muestra en la tabla siguiente.

Potencia	Elemento del campo	Potencia	Elemento del campo
α^1	α	α^9	$\alpha^2 + 1$
α^2	α^2	α^{10}	$\alpha^3 + \alpha$
α^3	α^3	α^{11}	$\alpha^3 + \alpha^2 + 1$
α^4	$\alpha^3 + 1$	α^{12}	$\alpha + 1$
α^5	$\alpha^3 + \alpha + 1$	α^{13}	$\alpha^2 + \alpha$
α^6	$\alpha^3 + \alpha^2 + \alpha + 1$	α^{14}	$\alpha^3 + \alpha^2$
α^7	$\alpha^2 + \alpha + 1$	α^{15}	1
α^8	$\alpha^3 + \alpha^2 + \alpha$		

Sea C el código $RS(15, 3)$ que resulta de $p(x)$. El siguiente es el polinomio generador $g(x)$ de C.

$$\begin{aligned} g(x) &= (x - \alpha)\ (x - \alpha^2)(x - \alpha^3)(x - \alpha^4)(x - \alpha^5)(x - \alpha^6) \\ &= ... \\ &= x^6 + \alpha^{12}x^5 + x^4 + \alpha^2x^3 + \alpha^7x^2 + \alpha^{11}x + \alpha^6 \end{aligned}$$

Para construir una de las palabras código en C, considere el polinomio en $F[x]$

$$b(x) = \alpha^8x^8 + \alpha^3x^7 + \alpha^7x^6 + \alpha^9x^5 + x^4 + \alpha^3x^3 + \alpha^6x^2 + \alpha^3x + \alpha^7.$$

Entonces

$$\begin{aligned} c(x) &= b(x)g(x) \\ &= ... \\ &= \alpha^8x^{14} + \alpha^{12}x^{13} + \alpha^3x^{12} + \alpha^8x^{11} + \alpha^{12}x^{10} + \alpha^3x^9 + \alpha^8x^8 + \alpha^{12}x^7 + \alpha^3x^6 + \alpha^4x^5 \\ &\quad + \alpha^4x^4 + \alpha^8x^3 + \alpha^{12}x^2 + \alpha^{11}x + \alpha^{13} \end{aligned}$$

es una de las palabras de código de C. Supongamos que se transmite $c(x)$ y se recibe el siguiente polinomio $r(x)$.

$$\begin{aligned} r(x) &= \alpha^8x^{14} + \alpha^{12}x^{13} + \alpha^3x^{12} + \alpha^8x^{11} + \alpha^{12}x^{10} + \alpha^3x^9 + \alpha^9x^8 + \alpha^5x^7 + \alpha^{13}x^6 + \alpha^4x^5 \\ &\quad + \alpha^4x^4 + \alpha^8x^3 + \alpha^{12}x^2 + \alpha^{11}x + \alpha^{13} \end{aligned}$$

Note que $r(x)$ contiene errores en las posiciones x^8, x^7, y x^6. Para corregir $r(x)$, ya que C corrige 3 errores, comenzamos calculando los primeros seis síndromes de $r(x)$. Enumeramos a continuación estos síndromes.

$$\begin{aligned} \mathrm{S}_1 = r(\alpha) = \cdots = 0 \\ \mathrm{S}_2 = r(\alpha^2) = \cdots = 0 \\ \mathrm{S}_3 = r(\alpha^3) = \cdots = \alpha^2 \\ \mathrm{S}_4 = r(\alpha^4) = \cdots = 1 \\ \mathrm{S}_5 = r(\alpha^5) = \cdots = 1 \\ \mathrm{S}_6 = r(\alpha^6) = \cdots = \alpha^{12} \end{aligned}$$

Estos síndromes producen el siguiente polinomio síndrome $S(z)$.

$$S(z) = \alpha^2 z^2 + z^3 + z^4 + \alpha^{12} z^5$$

Construyendo una tabla del algoritmo de Euclides para $a(z) = z^6$ y $b(z) = S(z)$ (omitiendo calculos numerosos) obtenemos lo siguiente.

FILA	**Q**	**R**	**V**
-1	$-$	z^6	0
0	$-$	$S(z)$	1
1	$\alpha^3 z + \alpha^6$	$\alpha^7 z^4 + \alpha^2 z^3 + \alpha^8 z^2$	$\alpha^3 z + \alpha^6$
2	$\alpha^5 z + \alpha^6$	$\alpha^4 z^3 + \alpha^3 z^2$	$\alpha^8 z^2 + \alpha^3 z + \alpha$
3	$\alpha^3 z + \alpha$	$\alpha^7 z^2$	$\alpha^{11} z^3 + \alpha^{10} z^2 + \alpha^3 z + \alpha^5$

Por lo tanto, $R(z) = \alpha^7 z^2$ y $V(z) = \alpha^{11} z^3 + \alpha^{10} z^2 + \alpha^3 z + \alpha^5$. Mediante la evaluación de $V(z)$ en las potencias sucesivas de α, podemos encontrar que las raíces de $V(z)$ son α^7, α^8 y α^9. Por lo tanto, las posiciones en $r(x)$ que contienen los errores son $x^{-7} = x^8$, $x^{-8} = x^7$, y $x^{-9} = x^6$. Para determinar los coeficientes de estos términos en el polinomio error, observamos en primer lugar que $V'(z) = \alpha^{11} z^2 + \alpha^3$. A continuación, se puede determinar los coeficientes de los términos del polinomio de error de la siguiente manera.

$$e_6 = \frac{R(\alpha^9)}{V'(\alpha^9)} = \frac{\alpha^{25}}{\alpha^{29} + \alpha^3} = \frac{\alpha^{10}}{\alpha^2} = \alpha^8$$
$$e_7 = \frac{R(\alpha^8)}{V'(\alpha^8)} = \frac{\alpha^{23}}{\alpha^{27} + \alpha^3} = \frac{\alpha^8}{\alpha^5} = \alpha^3$$
$$e_8 = \frac{R(\alpha^7)}{V'(\alpha^7)} = \frac{\alpha^{21}}{\alpha^{25} + \alpha^3} = \frac{\alpha^6}{\alpha} = \alpha^5$$

Por lo tanto, el error en $r(x)$ se puede expresar como el polinomio error $e(x) = \alpha^5 x^8 + \alpha^3 x^7 + \alpha^8 x^6$. Fácilmente se puede comprobar que formando $r(x) + e(x)$ se obtiene la palabra código $c(x)$.

4.6. Demostración de la corrección de errores en los códigos Reed-Solomon

En esta sección se verifica el esquema de corrección de errores resumidos e ilustrados en la sección anterior.

Sea F un campo de orden 2^n, y sea C un código $RS(2^n - 1,\ t)$ en $F[x]$. Supongamos que $c(x) \in C$ es transmitido y se recibe el polinomio $r(x) = c(x) + e(x)$ para algún polinomio error $e(x)$ distinto de cero en $F[x]$ de grado menor que $2^n - 1$. Se denotará este polinomio error como $e(x) = \sum_{j=0}^{m-1} e_j x^j$ con $m = 2^n - 1$ y coeficientes $e_j \in F$. Para determinar $e(x)$ a partir de $r(x)$ empezamos calculando los primeros $2t$ síndromes de $r(x)$. Denotamos estos síndromes como sigue.

$$S_i = r(\alpha^i) = e(\alpha^i) = \sum_{j=0}^{m-1} e_j \alpha^{ij} \quad para \quad i = 1, \ldots, 2t$$

A continuación, utilizamos estos síndromes para construir el polinomio síndrome de

$$S(z) = \sum_{i=0}^{2t-1} S_{i+1} z^i.$$

Notemos que

$$S(z) = \sum_{i=0}^{2t-1} \sum_{j=0}^{m-1} e_j \alpha^{(i+1)j} z^i = \sum_{j=0}^{m-1} e_j \alpha^j \sum_{i=0}^{2t-1} \alpha^{ij} z^i.$$

Sea M el conjunto de enteros que corresponden a las posiciones de error en $r(x)$. Es decir, sea $M = \{\ \mathrm{j} \leq m - 1 \mid e_j \neq 0\ \}$. Note además que

$$S(z) = \sum_{j \in M} e_j \alpha^j \sum_{i=0}^{2t-1} \alpha^{ij} z^i = \sum_{j \in M} e_j \alpha^j \left(\frac{1 - \alpha^{j(2t)} z^{2t}}{1 - \alpha^j z} \right)$$

$$= \sum_{j \in M} \frac{e_j \alpha^j}{1 - \alpha^j z} - \sum_{j \in M} \frac{e_j \alpha^{j(2t+1)} z^{2t}}{1 - \alpha^j z}$$

Por lo tanto, para los polinomios

$$R(z) = \sum_{j \in M} e_j \alpha^j \prod_{\substack{i \in M \\ i \neq j}} (1 - \alpha^i z),$$
$$U(z) = \sum_{j \in M} e_j \alpha^{j(2t+1)} \prod_{\substack{i \in M \\ i \neq j}} (1 - \alpha^i z), \quad y$$
$$V(z) = \prod_{i \in M} (1 - \alpha^i z),$$

resulta que

$$S(z) = \frac{R(z)}{V(z)} + \frac{U(z) z^{2t}}{V(z)},$$

o, equivalentemente,

$$U(z) z^{2t} + V(z) S(z) = R(z).$$

Esta última ecuación es llamada *ecuación fundamental*. En esta ecuación, $V(z)$ se llama el polinomio *localizador de error*, $R(z)$ se llama el polinomio *evaluador de error*, y $U(z)$ se llama el polinomio *coevaluador de error*. Note que

$$(U(z),\ V(z)) = (R(z),\ V(z)) = 1.$$

Consideremos ahora la forma de determinar $V(z)$, $R(z)$, y $U(z)$. Como se verá, estos polinomios son las entradas de la tabla del algoritmo de Euclides para un $a(z) = z^{2t}$ y $b(z) = S(z)$ en la primera fila j para la que $deg(r_j) < t$. Los siguientes resultados verifican esto. Para mayor comodidad, en estos resultados vamos a suprimir la variable z siempre que sea posible.

Teorema. Supongamos que $VS + Uz^{2t} = R$ para algún polinomio síndrome S, y sean V_0, U_0, y R_0 polinomios que satisfacen que

$$V_0 S + U_0 z^{2t} = R_0\,; \quad deg(V_0) \leq t,\ deg(U_0) < t,\ deg(R_0) < t.$$

Entonces existe un polinomio h $\in F[z]$ tal que $V_0 = hV$, $U_0 = hU$, y $R_0 = hR$. Si es cierto también que $(V_0,\ U_0) = 1$, entonces h es constante.

Demostración.

Nótese en primer lugar que, como $VS + Uz^{2t} = R$ y $V_0S + U_0z^{2t} = R_0$, se sigue que

$$V_0VS + V_0Uz^{2t} = V_0R$$

y

$$VV_0S + VU_0z^{2t} = VR_0.$$

Por lo tanto, por sustracción,

$$(V_0U - VU_0)z^{2t} = V_0R - VR_0.$$

Por un argumento de grado, podemos ver que ambos lados de la ecuación anterior deben ser igual a 0. Por lo tanto,

$$V_0U - VU_0 = V_0R - VR_0 = 0.$$

Ya que $(V,\ U) = 1$, entonces deben existir polinomios α, $\beta \in F[z]$ para los que $\alpha V + \beta U = 1$. Por lo tanto,

$$V_0\alpha V + V_0\beta U = V_0.$$

Pero ya que $V_0U = VU_0$, entonces

$$V_0\alpha V + V\beta U_0 = V_0,$$

o, equivalentemente,

$$(V_0\alpha + U_0\beta)V = V_0.$$

Ahora, sea

$$h = V_0\alpha + U_0\beta.$$

Entonces $hV = V_0$. También, $hVU = V_0U = VU_0$ implica $hU = U_0$, y $hVR = V_0R = VR_0$ implica $hR = R_0$. Por último, ya que h debe dividir a V_0 y U_0, si $(V_0,\ U_0) = 1$, entonces h debe ser constante.

∎

Teorema. Supongamos $a = z^{2t}$ y $b = S$ para algún polinomio síndrome S. En la tabla del algoritmo Euclidiano para a y b, sea j la primera fila para la que $deg(r_j) < t$. Define $R_0 = r_j,\ U_0 = u_j,\ V_0 = v_j$. Entonces $R_0,\ U_0$, y V_0 satisfacen todas las condiciones del teorema anterior.

Demostración.

De la ecuación (1.5) sabesmos que $r_j = u_jz^{2t} + v_jS$. Por lo tanto $R_0 = U_0z^{2t} + V_0S$. Además, puesto que $R_0 = r_j$ y $deg(r_j) < t$, sabemos que $deg(R_0) < t$.

Ahora, porque $deg(v_{j-1}) < deg(v_j) = deg(V_0)$ y $deg(r_{j-1}) > deg(v_r) = deg(R_0)$, resulta que $deg(v_{j-1}R_0) < deg(V_0r_{j-1})$. Pero por la ecuación (1.7), sabemos que $R_0v_{j-1} - r_{j-1}V_0 = a = z^{2t}$. Por lo tanto, $deg(r_{j-1}V_0) \leq 2t$, y como $deg(r_{j-1}) \geq t$, resulta que $deg(V_0) \leq t$.

También, como $deg(u_{j-1}) < deg(u_j) = deg(U_0)$, debe ser el caso de que $deg(R_0u_{j-1}) < deg(r_{j-1}U_0)$. Pero por la ecuación (1.6) seabemos que $R_0u_{j-1} - r_{j-1}U_0 = b = S$. Por lo tanto, $deg(U_0r_{j-1}) < 2t$, y puesto que $\deg(r_{j-1}) \geq t$, resulta que $deg(U_0) < t$.

Queda por demostrar sólo que $(V_0, U_0) = 1$. Pero por la ecuación (1.8) sabemos que $u_{j-1}v_j - u_jv_{j-1} = 1$. Por lo tanto $u_{j-1}V_0 - U_0v_{j-1} = 1$, y $(V_0, U_0) = 1$.

∎

En resumen, para un polinomio síndrome $S(z)$, para determinar los polinomios localizador de error, evaluador de error, y coevaluador de error $V(z)$, $R(z)$, y $U(z)$, se construye la tabla del algoritmo de Euclides para $a(z) = z^{2t}$ y $b(z) = S(z)$. En la primera fila j para la cual $deg(r_j) < t$, entonces $r_j = R_0 = hR(z)$, $u_j = U_0 = hU(z)$, $v_j = V_0 = hV(z)$. Pero puesto que $(V_0, U_0) = 1$, entonces $h = 1$. Por lo tanto, $r_j = R(z)$, $u_j = U(z)$, y $v_j = V(z)$. Al encontrar las raíces de $V(z)$, podemos determinar la

ubicación de los errores en el polinomio recibido como se describió anteriormente. Para encontrar los coeficientes de los términos del polinomio de error, note que puesto que

$$V(z) = \prod_{i \in M} (1 - \alpha^i z),$$

entonces

$$V'(z) = \sum_{j \in M} -\alpha^j \prod_{\substack{i \in M \\ i \neq j}} (1 - \alpha^i z).$$

Y recordemos que ya sabemos que

$$R(z) = \sum_{j \in M} e_j \alpha^j \prod_{\substack{i \in M \\ i \neq j}} (1 - \alpha^i z).$$

Mediante la evaluación de los polinomios anteriores en α^{-j}, se obtiene lo siguiente.

$$V'(\alpha^{-j}) = -\alpha^j \prod_{\substack{i \in M \\ i \neq j}} (1 - \alpha^{i-j})$$

$$R(\alpha^{-j}) = e_j \alpha^j \prod_{\substack{i \in M \\ i \neq j}} (1 - \alpha^{i-j})$$

Por lo tanto, $\dfrac{R(\alpha^{-j})}{V'(\alpha^{-j})} = e_j$ revela el coeficiente de x^j en el polinomio error.

4.7. Códigos Reed-Solomon binarios

Se indicó anteriormeente que palabras código de los códigos Reed-Solomon se transmiten como vectores binarios, pero no hemos mencionado todavía la forma concreta en que las palabras código se transmiten. A continuación vamos a discutir la forma en que las palabras código de los códigos Reed-Solomon se transmiten.

Considere la palabra código $c(x)$ en el código Reed-Solomon C del ejemplo 1. Para convertir $c(x)$ a un vector binario, comenzamos haciendo un listado de los términos en $c(x)$ de la siguiente manera, con potencias cada vez mayores de x.

$$\begin{aligned} c(x) = {} & \alpha^{13} + \alpha^{11}x + \alpha^{12}x^2 + \alpha^{8}x^3 + \alpha^{4}x^4 + \alpha^{4}x^5 + \alpha^{3}x^6 + \alpha^{12}x^7 \\ & + \alpha^{8}x^8 + \alpha^{3}x^9 + \alpha^{12}x^{10} + \alpha^{8}x^{11} + \alpha^{3}x^{12} + \alpha^{12}x^{13} + \alpha^{8}x^{14} \end{aligned}$$

A continuación, se escriben los coeficientes $c(x)$ de la siguiente manera, usando la tabla en el ejemplo 1 para expresar cada coeficiente como un polinomio en α de grado menor que el grado de $p(x)$ con potencias cada vez mayores de α.

$$\begin{aligned} c(x) = {} & (\alpha + \alpha^2) + (1 + \alpha^2 + \alpha^3)x + (1 + \alpha)x^2 + (\alpha + \alpha^2 + \alpha^3)x^3 \\ & + (1 + \alpha^3)x^4 + (1 + \alpha^3)x^5 + (\alpha^3)x^6 + (1 + \alpha)x^7 \\ & + (\alpha + \alpha^2 + \alpha^3)x^8 + (\alpha^3)x^9 + (1 + \alpha)x^{10} + (\alpha + \alpha^2 + \alpha^3)x^{11} \\ & + (\alpha^3)x^{12} + (1 + \alpha)x^{13} + (\alpha + \alpha^2 + \alpha^3)x^{14} \end{aligned}$$

Por último, queremos expresar cada uno de estos coeficientes de $c(x)$ como vectores binarios de longitud de cuatro listando en orden los coeficientes binarios de α. Por ejemplo, queremos expresar el primer coeficiente $0 + 1\alpha + 1\alpha^2 + 0\alpha^3$ de $c(x)$ como el vector (0110). Usando este método, se convertiría toda la palabra código $c(x)$ en un vector binario de longitud de 60 posiciones juntando las listas de vectores binarios de longitud de cuatro, incluyendo la de cuatro ceros para todos los términos que podrían estar presentes en $c(x)$, pero tienen coeficiente 0. Es decir, se convertiría $c(x)$ en el siguiente vector binario de longitud de 60.

(011010111100011110011001000111000111000111000111000111000111).

Es claro que por el hecho de que las palabras código Reed-Solomon se convierten en vectores binarios de esta manera son ideales para la corrección de errores en rafagas. En concreto, cuatro errores en su equivalente binario de $c(x)$ construido anteriormente podría representar un solo error en $c(x)$.

4.8. Códigos Reed-Solomon con *GAP*

Esta ves hemos decidido utilizar *GAP* para programar los códigos Reed-Solomon debido a que en esta ocasión trabajaremos con campos finitos y *GAP* nos mostró a la hora de estar programando ser mas veloz al realizar operaciones sobre campos finitos que *Mathematica* y solo al final utilizaremos *Mathematica* para mostrar los resultados de una simulación del envío de una imagen codificada con un código $RS(15, 5)$ y una sin codificar para comparar los resultados. En esta ocasión se verá con mas detalle la construcción del código y su decodificación ya que estos códigos son mas complejos que lo anteriores.

Construcción del código:

Primero que nada necesitamos el campo finito de 16 elementos y lo obtenemos de la siguiente manera:

```
F:=GF(16);
```

para construir todos los polinomios en $F[x]$ de grado menor que 5 necesitamos una función auxiliar que convierta una sucesión de elementos de F en un su correspondiente polinomio

```
pol:=function ( p )
    local  s, m, f, j, x;
    s := p;
    m := Size( s );
    f := 0;
    x := Indeterminate(GF(16));
    for j  in [ 1 .. m ]  do
        f := f + s[j] * x ^ (j - 1);
    od;
    return f;
```

```
end;
```

Para codificar necesitamos la transformada discreta de Fourier que la escribimos como:

```
Ft:=function( p, t )
    local  f, c, j, x, ft;
    f := p;
    c := t;
    x := Indeterminate(GF(16));
    ft := 0;
    for j  in [ 1 .. 15 ]  do
        ft := ft + Value( f, c ^ (j - 1) ) * x ^ (j - 1);
    od;
    return ft;
end;
```

ahora para construir el código escribimos:

```
tu := Tuples( F, 5 );;

p := [  ];;

for i  in [ 1 .. Size( tu ) ]  do
        p[i] := pol( tu[i] );
    od;

C := [  ];;

for i  in [ 1 .. Size( p ) ]  do
        C[i] := Ft( p[i], Z(16) );
    od;
```

en donde al finalizar C es nuestro código RS(15, 5). Note que este código tiene 1048576 elementos por lo que realizar los anterior puede llevar algún tiempo considerable.

Decodificación

Primero y solo por simplicidad a la hora de escribir renombramos dos funciones utiles de *GAP.*

```
deg :=  DegreeOfLaurentPolynomial;

coef:= CoefficientsOfUnivariatePolynomial;
```

Ahora recordemos que lo primero que se tiene que hacer es calcular el síndrome del polinomio recibido y para eso construimos la siguiente función que lo calcula:

```
sdr:=function ( p, t, k )
    local  f, c, m, sd, j;
    f := p;
    c := t;
    m := k;
    sd := 0*Z(16);
    for j  in [ 1 .. m ]  do
        sd := sd + Value( f, c ^ j ) * x ^ (j - 1);
    od;
    return sd;
end;
```

ahora necesitamos un programa que calcule la tabla del algoritmo de Euclides y lo realizamos como sigue:

primero programamos el proceso de inducción:

```
ind := function( p )
    local  t, t1, c;
    t := p;
    t1 := [  ];
    if deg( t[4] ) > 0  then
        c := EuclideanQuotient( t[2], t[4] );
    fi;
    if
     deg( t[4] ) = 0 and t[4] = 0 * Z( 16 )  then
        c := 0 * Z( 16 );
    else
        c := EuclideanQuotient( t[2], t[4] * x ^ 0 );
    fi;
    t1 := [ t[3], t[4], t[1] - c * t[3], t[2] - c * t[4] ];
    return t1;
```

```
end;
```

Después escribimos una función que utilice ese proceso de inducción hasta que cumpla con la condición que queremos.

```
mcd := function( p, q, k )
    local  t, m;
    t := [ 0 * Z( 16 ), p, Z( 16 ) ^ 0, q ];
    m := k;
    repeat
        t := ind( t );
    until deg( t[4] ) < m;
    return t;
end;
```

como necesitamos calcular la derivada de un polinomio y encontrar sus raíces escribimos las siguientes funciones que lo realizan.

```
der:=function ( p )
    local  f, s, m, d, j;
    f := p;
    s := coef( f );
    m := Size( s );
    d := [  ];
    for j  in [ 1 .. m - 1 ]  do
        d[j] := j * Z( 16 ) ^ 0 * s[(j + 1)];
    od;
    return pol( d );
end;

root:=function ( p )
    local  f, s, k, a, j;
    f := p;
    s := [  ];
    a := Z( 16 );
    k := 0;
    for j  in [ 1 .. 15 ]  do
        if Value( f, a ^ j ) = 0 * a  then
            ;
            k := k + 1;
            s[k] := 15-j;
        fi;
    od;
    return s;
```

```
end;
```

Ahora si podemos escribir la función que calcula los errores y lo hacemos de la siguiente manera

```
ero:=function ( u, h, t )
    local  v, r, s, e, a, j;
    v := u;
    r := h;
    s := t;
    e := [  ];
    a := Z( 16 );
    for j  in [ 0 .. 14 ]  do
        if j in s  then
            e[j+1] := Value( r, a ^( 15-j ))
              * Value( der( v ), a ^( 15- j) ) ^( -1);
        else
            e[j+1] := 0 * a;
        fi;
    od;
    return e;
end;
```

por ultimo escribimos una función que realice todo el proceso de decodificación utilizando las funciones escritas anteriormente.

```
deco := function ( p )
    local  f, sd, m, v, r, s, e, a;
    f := p;
    a := Z( 16 );
    sd := sdr( f, a, 10 );
    if sd = 0 * Z( 16 ) * x  then
        e := [ sd ];
    else
        m := mcd( x ^ 10, sd, 5 );
        v := m[3];
        r := m[4];
        s := root( v );
        e := ero( v, r, s );
    fi;
    return pol( e );
end;
```

Ahora para simular la transmisión de una imagen necesitamos código Reed-Solomon binarios para hacer esto construimos las siguientes funciones en *GAP*

```
f416:=function ( x )
    local  a, s, gf, i;
    s := x;
    a := Z( 16 );
    gf := a * 0;
    for i  in [ 1 .. 4 ]  do
        if s[i] = 0  then
            gf := gf + a * 0;
        else
            ;
            gf := gf + a ^ (i - 1);
        fi;
    od;
    return gf;
end;
```

Esta función transforma un vector binario de cuatro posiciones en un elemento del campo F previamente definido. Para realizar la operación inversa podemos construir una tabla aplicando la función f416 a los elementos de t

```
t:=[ [ 0, 0, 0, 0 ], [ 0, 0, 0, 1 ], [ 0, 0, 1, 0 ],
  [ 0, 0, 1, 1 ], [ 0, 1, 0, 0 ], [ 0, 1, 0, 1 ], [ 0, 1, 1, 0 ],
  [ 0, 1, 1, 1 ], [ 1, 0, 0, 0 ], [ 1, 0, 0, 1 ], [ 1, 0, 1, 0 ],
  [ 1, 0, 1, 1 ], [ 1, 1, 0, 0 ], [ 1, 1, 0, 1 ], [ 1, 1, 1, 0 ],
  [ 1, 1, 1, 1 ] ];
```

y así obtenemos

```
gf16:=[  0*Z(2),  Z(2^4)^3,  Z(2^4)^2,  Z(2^4)^6,  Z(2^4),  Z(2^4)^9,
Z(2^2), Z(2^4)^11, Z(2)^0, Z(2^4)^14, Z(2^4)^8, Z(2^4)^13, Z(2^4)^4,
Z(2^4)^7, Z(2^2)^2, Z(2^4)^12 ];
```

entonces la función inversa de f416 es la siguiente.

```
if416:=function ( x )
    local  gf, k, v;
    gf := x;
    k := 0;
    repeat
```

```
        k := k + 1;
    until gf = gf16[k];
    return t[k];
end;
```

También vamos a necesitar la siguiente función auxiliar que parte una vector en pedazos en vectores de cuatro posiciones

```
part:=function ( x )
    local  s, p, i, j, z;
    s := x;
    p := [  ];
    z := (Size( s )/4)-1;
    for i  in [ 0 .. z ]  do
        ;
        p[i + 1] := [  ];
        for j  in [ 1 .. 4 ]  do
            ;
            Add( p[i + 1], s[4 * i + j] );
        od;
    od;
    return p;
end;
```

Ahora si escribimos la función que nos transforma un vector binario de 60 posiciones en su correspondiente polinomio sobre $F[x]$.

```
cpol:=function ( x )
    local  s, g, i, z;
    s := part( x );
    g := [  ];
    z := Size( s );
    for i  in [ 1 .. z ]  do
        Add( g, f416( s[i] ) );
    od;
    return pol( g );
end;
```

Y su inversa es la siguiente

```
icpol:=function ( x )
    local  p, s, i, z;
    p := coef( x );
    s := [  ];
```

```
    z := Size( p );
    if p = [  ]  then
        s := [ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ];
    else
        for i  in [ 1 .. z ]  do
            Add( s, if416( p[i] ) );
        od;
    fi;
    return Flat( s );
end;
```

Para simular la transmisión de una imagen necesitamos las siguientes funciones

```
burst:=function ( x )
    local  a, b, i, s, m;
    a := x;
    b := Random( [ 1 .. 60 ] );
    s := [  ];
    for i  in [ 1 .. b ]  do
        s[i] := 0;
    od;
    for i  in [ b + 1 .. b + a ]  do
        if i <= 60  then
            if Random( [ 1 .. 6 ] ) in [ 1 .. 5 ]  then
                s[i] := 1;
            else
                s[i] := 0;
            fi;
        else
            m := i mod 60;
            if Random( [ 1 .. 6 ] ) in [ 1 .. 5 ]  then
                s[m] := 1;
            else
                s[m] := 0;
            fi;
        fi;
    od;
    if b + a < 60  then
        for i  in [ b + a + 1 .. 60 ]  do
            s[i] := 0;
        od;
    fi;
    return s;
end;
```

Con la que construimos simulamos ráfagas de errores por ejemplo si escribimos `burst(17)` nos devuelve

```
[ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1,
1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ]
```

En nuestro caso construiremos arreglo al que llamaremos "errores" de 200 × 250 en donde cada elemento de este arreglo es un vector de largo 60 construido utilizando la función burst, en seguida mostramos una parte.

```
errores:=[ [ [ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1,
1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0 ], [ 0, 0, 0,
0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  1,  1,  1,  0,  1,  1,
1,  1,  1,  0,  1,  1,  1,  1,  1,  1,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 ], [ 1,  1,  1,  1,  0,  1,  0,
0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  1,  1,  1,  1,
1,  1,  1,  1,  1,  1,  1 ], [ 0,  0,  1,  1,  1,  1,  0,  1,  1,  1,  1,  1,  1,  1,  1,
1,  1,  1,  1,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 ],
[ 1,  1,  1,  1,  1,  1,  1,  0,  1,  1,  1,  1,  1,  1,  0,  0,  0,  0,  0,  0,  0,  0,
0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  1 ],...]];
```

Ahora introducimos los colores que ocupara nuestra imagen en nuestro caso son los siguientes y debido a que utilizaremos 6320 colores diferentes no los podemos escribir todos en este documento solo pondremos algunos para ilustrar como se vería.

```
colores:=[[0,0,0],[0,0,2],[0,0,4],[0,0,5],[0,1,0],[0,1,3],[0,1,5],
[0,1,6],[0,2,0],[0,2,1],[0,2,3],[0,2,5],[0,2,7],[0,2,9],[0,2,14],
[0,3,0],[0,3,2],[0,3,4],[0,3,6],[0,3,7],[0,4,0],[0,4,3],[0,4,5],
[0,4,7],[0,5,0],[0,5,1],[0,5,11],[0,6,0],[0,6,2],[0,6,4],[0,6,12],
[0,7,13],[0,8,0],[0,8,3],[0,8,7],[0,9,1],[0,9,8],[0,10,14],[1,0,0],
[1,0,2],[1,0,5],[1,0,6],[1,1,0],[1,1,1],[1,1,3],[1,2,0],[1,2,4],
[1,2,6],[1,3,0],[1,3,2],[1,5,4],[1,5,6],[1,5,8],[1,6,0],[1,6,2],
[1,6,9],[1,6,10],[1,7,3],[1,7,5],[1,8,1],[1,11,3],[2,0,1],[2,0,3],
[2,0,5],[2,0,6],[2,0,11],[2,0,13],[2,1,0],[2,1,6],[2,1,7],[2,1,9],
[2,2,0],[2,2,2],[2,2,4],[2,3,0],[2,3,5],[2,3,7],[2,3,8],[2,4,0],
[2,4,1],[2,4,3],[2,5,0],[2,5,10],[2,5,12],[2,6,5],[2,6,9],[2,7,0],
[2,7,3],[2,11,8],[2,11,10],[3,0,0],[3,0,2],[3,1,2],[3,1,4],[3,2,0],
[3,2,7],[3,2,8],[3,2,10],[3,3,1],[3,3,3],[3,3,5],[3,4,0],[3,4,6],
```

```
[3,4,8],[3,4,9],[3,5,0],[3,5,2],[3,5,4],[3,6,0],[3,7,6],[3,7,8],
[3,7,10],[3,8,1],[3,8,4],[3,8,11],...];
```

De manera similar introducimos la imagen que utilizaremos

```
data:=[[[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0],
[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0],
[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0],
[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0],
[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0],
[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0],
[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0],
[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0],
[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0],
[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0],
[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0],
[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0],
[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0],
[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0],
[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0],
[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0],
[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0],
[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0],
[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0],
[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0],[2,2,2],
[3,3,3],[4,0,0],[4,0,0],[3,0,0],[5,1,0],[6,2,1],[6,2,0],[6,2,0],
[6,2,0],[6,2,0],[6,2,0],[6,2,0],[6,2,0],[6,2,0],[6,2,0],[6,2,0],
[6,2,0],[5,1,0],[6,2,0],[6,2,0],[6,2,1],[4,0,0],[4,0,0],[4,0,0],
[5,1,0],[6,2,0],[6,2,0],[6,2,0],[6,2,0],[6,2,0],[6,2,0],[6,2,0],
[6,2,0],[6,2,0],[6,2,0],[6,2,0],[6,2,0],[6,2,0],[6,2,0],[6,2,0],
[6,2,0],[6,2,0],[6,2,0],[6,2,0],[6,2,0],[6,2,0],[6,2,0],[6,2,0],
[6,2,0],[6,2,0],[6,2,0],[6,2,0],[6,2,0],[6,2,0],[6,2,0],[6,2,0],
[6,2,0],[5,1,0],[4,0,0],[2,0,1],[2,0,1],[2,0,1],[2,0,1],[4,0,1],
[4,0,1],[4,0,0],[4,0,0],[4,0,0],[4,0,0],[4,0,0],[4,0,0],[4,0,0],
[5,1,0],[4,0,1],[4,0,1],[4,0,1],[4,0,0],[4,0,0],[4,0,0],[4,0,0],
[4,0,0],[4,0,0],[4,0,0],[4,0,0],[4,0,0],[4,0,0],[4,0,0],[4,0,0],
[4,0,0],[4,0,0],[4,0,0]],...]];
```

Tampoco podemos escribir aquí todos los elementos de la imagen ya que es un arreglo de 200×250.

La siguiente función es la que asigna a cada color de nuestra imagen un elemento de nuestro código

```
f1:=function ( x )
    local  a, i;
```

```
    a := x;
    i := 0;
    repeat
        i := i + 1;
    until a = colores[i];
    return bC[i];
end;
```

y escribimos

```
Y:=data;;for i in [1..200] do
    for j in [1..250] do
    Y[i][j]:=f1(data[i][j]);od;od;
```

Y la siguiente función devuelve a cada elemento del código el color que le corresponde si el vector recibido no esta en el código entonces le asigna el un color fijo.

```
f2:=function ( x )
    local  c,a, i;
    a := x;
    i := 0;
    if a in bC then repeat
         i:=i+1; until
     a=bC[i];
         return colores[i]; else
         return colores[1];fi;end;
```

Por ultimo para simular la transmisión escribimos lo siguiente

```
Te:=Y;;for i in [1..200] do
    for j in [1..250] do
    Te[i][j]:=(Y[i][j]+errores[i][j]) mod 2;od;od;

D:=Te;;for i in [1..200] do
    for j in [1..250] do
    D[i][j]:=icpol(deco(cpol(Te[i][j]))+cpol(Te[i][j]));od;od;

O:=D;; for i in [1..200] do
    for j in [1..250] do
    O[i][j]:=f2(D[i][j]);od;od;
```

Transformando la matriz O en su correspondiente imagen obtenemos la siguiente imagen que contiene cero errores con respecto a la original

Si hubiéramos realizado la transmisión de la imagen sin haberla codificado hubiéramos obtenido la siguiente imagen que contiene 29607 errores

5. Códigos Convolutivos

Los códigos que consideramos en este capítulo son bastante diferentes a los de los capítulos anteriores. No son códigos de bloque, es decir, las palabras no tienen longitud constante. Aunque hay analogías y conexiones con los códigos de bloque, hay una gran diferencia, a saber, la teoría matemática de los códigos convolutivos no está bien desarrollada. Esta es una de las razones que los matemáticos les resulta difícil llegar a interesarse también por estos códigos.

Los códigos convolutivos son comúnmente especificados por tres parámetros; (n, k, m).

n = número de bits de salida
k = número de bits de entrada
m = número de registradores de memoria

La cantidad k/n llama la *razón del código*, es una medida de la eficiencia del código. A menudo, los fabricantes de chips de códigos convolutivos especifican el código con los parámetros (n, k, L), la cantidad de L se denomina *restricción de longitud* del código y se define por

$$L = k(m - 1)$$

Todas las introducciónes a la codificación convolutiva parecen usar el mismo ejemplo. Existen numerosos ejemplos pero sin embargo vamos a utilizar los ejemplos canónicos.

En la Figura 1 se muestra el dispositivo de codificación de nuestro código. Los tres cuadros son elementos de almacenamiento (*registradores de memoria*) que pueden estar en uno de los dos posibles estados que denotaremos por 0 y 1. El sistema se rige por un reloj externo que produce una señal cada t_0 segundos (por simplicidad elegimos nuestra unidad de tiempo tal que $t_0 = 1$). La estructura de un código convolutivo es fácil de obtener de sus parámetros. Primero dibuja m cuadros que representan los registradores de memoria. A continuación, dibuje n sumadores mod 2 para representar los n bits de de salida. La selección de cuales bits se van a sumar para obtener el bit de de salida se llama el polinomio generador g para ese bit de salida. Por ejemplo, en la Figura 1 la

salida T_0 tiene polinomio generador (1, 0, 1) y la salida T_1 tiene polinomio generador (1, 1 ,1).

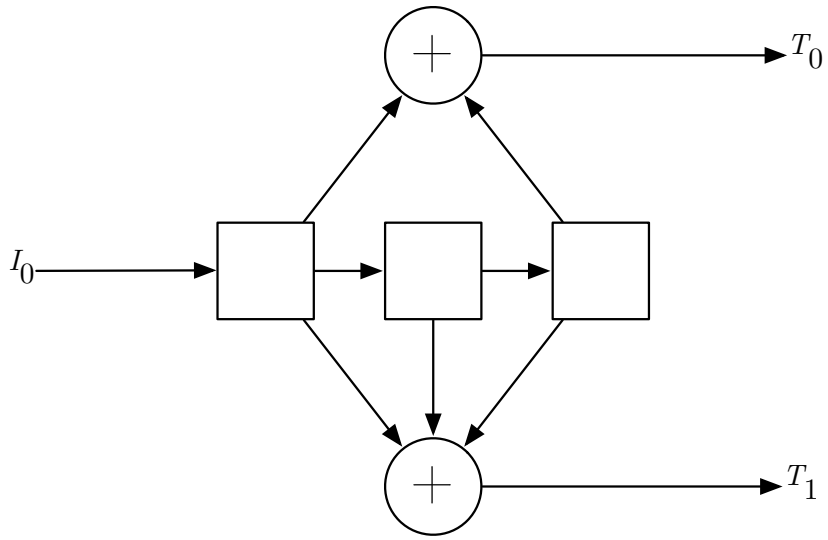


Figura 1. Código convolutivo (2, 1, 3).

El efecto de esta señal es que el contenido del registrador de memoria se mueve en la dirección de las flechas al siguiente elemento de este llamado *registrador de cambio* (en la Figura 1 el registrador de cambio esta compuesto por tres registradores de memoria). Los elementos $\oplus$ indican *sumadores* mod 2. Por cada pulso del reloj, vemos que el contenido del primer y tercer registrador de memoria se suman y luego salen del codificador a través de la secuencia T_0. La información que se procesará entra por la izquierda como una secuencia I_0, Observe que el primer registrador de memoria es en realidad superfluo, ya que sólo sirve para dividir la secuencia de entrada en tres direcciones. El segundo y tercer registrador de memoria muestran la diferencia esencial con los códigos de bloque. Son elementos de memoria en las que se ve que en el tiempo t las señales de entrada para $t-1$ y $t-2$ están todavía disponibles. La salida depende de estos tres símbolos de entrada. En la práctica la secuencia de salida de T_0 y T_1 se entrelazan para producir una sola secuencia de salida. Por lo tanto, si empezamos con (0 0 0) en los registradores de memoria y se tiene como mensaje una secuencia que consiste de 1 y seguida de 0s, vemos que el registrador primero cambia a (1 0 0) y produce la salida de (1 1); luego se mueve a (0 1 0) con una salida de (0 1); posteriormente (0 0 1) con una salida de (1 1) y luego vuelve al estado original y una secuencia de 0s. Veamos con mas detalle como funciona este procedimiento de codificación.

5.1. Estados de un código

El código (2, 1, 4) de la Figura 2 tiene restricción de longitud 3. Los registradores de memoria sombreados abajo mantienen estos bits. El registrador sin sombra tiene el bit de entrada. Esto significa que 3 bits u 8 diferentes combinaciones de estos bits pueden estar presentes en estos registros de memoria sombreados. Estas ocho combinaciones diferentes determinan que salida se obtiene para V_1 y V_2.

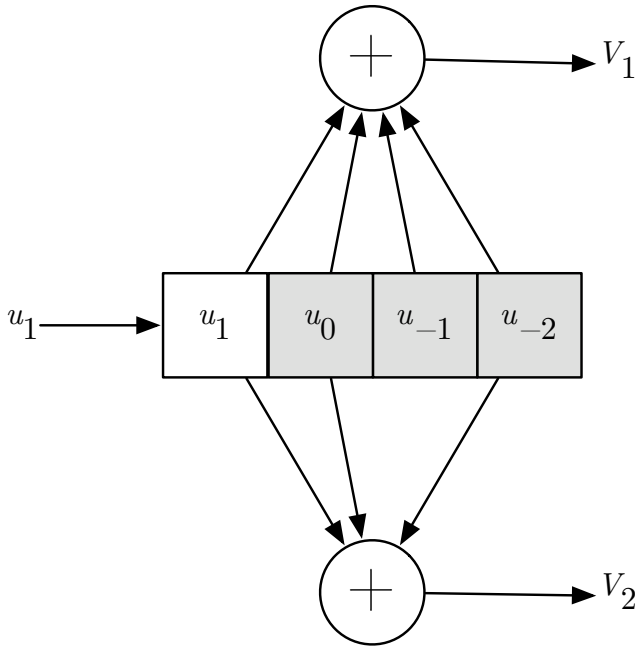


Figura 2. Los estados de un código de indican lo que hay en los registradores de memoria.

El número de combinaciones de bits en los registradores de la sombreados se llaman los estados del código y se definen por

$$\text{Número de estados} = 2^L$$

donde L es la restricción de longitud del código.

Los 8 estados del código de la Figura 2 son: 000, 001, 010, 011, 100, 101, 110, 111.

5.2. Codificación de una secuencia de entrada

Un codificador convolutivo se llama así porque se realiza una convolución de la secuencia de entrada con las *respuestas al impulso* del codificador:

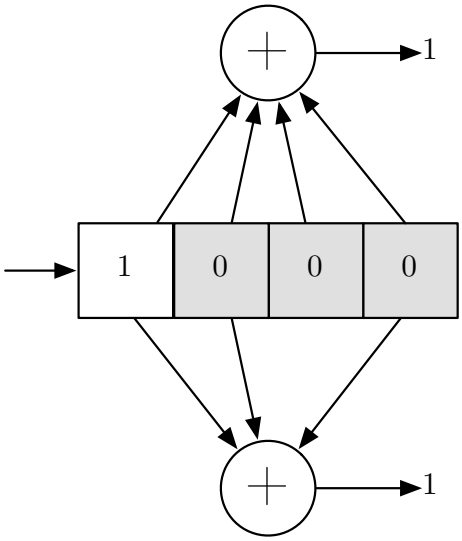
$$v = u * g$$

o en una forma más genérica

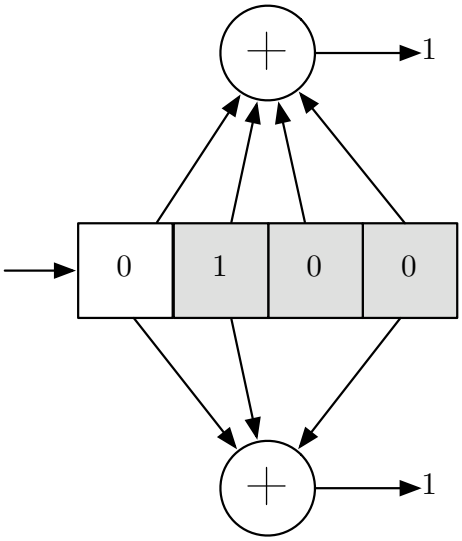
$$v_i^j = \sum_{k=0}^{\infty} u_{i-k} g_k^j$$

donde v_i^j es el i bit de salida del codificador j, u_{i-k} es el bit de entrada, y g_k^j es el término k en el polinomio j.

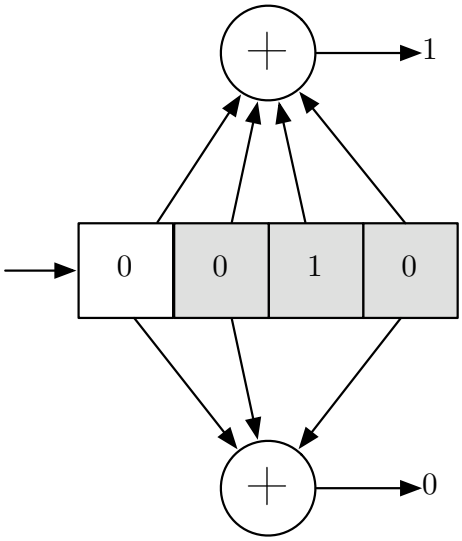
Vamos a codificar la secuencia de dos bits 10 con el código (2, 1, 4) de la Figura 2 y ver cómo funciona el proceso.



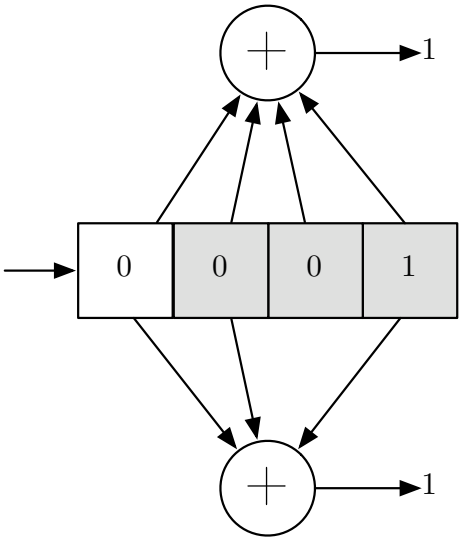
a) $t = 0$, Estado = 000
Bit de entrada = 1, Bits de salida = 11



b) $t = 1$, Estado = 100
Bit de entrada = 0, Bits de salida = 11



c) $t = 2$, Estado = 010
Bit de entrada = 0, Bits de salida = 10



d) $t = 3$, Estado = 001
Bit de entrada = 0, Bits de salida = 11

En el tiempo 4, el bit 1 de entrada ha pasado por completo por el codificador y el codificador se ha vuelto a un estado de puros ceros, listo para la siguiente secuencia.

Lo que se acaba de producir se llama la *respuesta al impulso* de este codificador. El bit 1 tiene una respuesta de 11 11 10 11. El bit 0 de manera análoga tiene una respuesta al impulso de 00 00 00 00.

Una descripción matemática de este procedimiento de codificación se puede dar de la siguiente manera. Escribimos la secuencia de entrada i_0, i_1, i_2, ... por la serie de potencias formal $I_0(x) := i_0 + i_1x + i_2x^2 + \cdots$ con coeficientes en $\mathbb{F}_2$. Del mismo modo escribimos las salidas de V_1 y V_2 como series de potencias $V_1(x)$ y $V_2(x)$ respectivamente. Sincronizamos el tiempo de tal manera que la primera entrada corresponde a la primera salida. Entonces está claro que

$$V_1(x) = (1 + x + x^2 + x^3)I_0(x),$$
$$V_2(x) = (1 + x + x^3)I_0(x).$$

El entrelazamiento de V_1 y V_2 entonces se describe por

$$T(x) = V_1(x^2) + xV_2(x^2).$$

donde $T(x)$ produce el conjunto de las posibles secuencias de salida.

De una manera completamente análoga para los códigos convolutivos de razón $1/n$. Tenemos una secuencia de información dada por la serie $I_0(x)$. Hay n secuencias de salida: $V_1(x)$, $V_2(x)$, . . . , $V_n(x)$, donde cada secuencia codificada $V_i(x)$ se obtiene multiplicando $I_0(x)$ por un polinomio generador $g_i(x)$. La secuencia transmitida es entonces

$$T(x) = \sum_{i=1}^{n} x^{i-1}V_i(x^n).$$

Es obvio que la elección de los polinomios generadores $g_i(x)$ determina si el código es bueno o malo, cualquier significado que nosotros le demos a esto. Vamos ahora a describir una situación que es obviamente mala. Supongamos que la secuencia entrada $I_0(x)$ contiene un número infinito de 1s pero la correspondiente secuencia de salida $T(x)$ sólo tiene un número finito de 1s. Si el canal de transmisión accidentalmente comete errores en las posiciones donde están los 1s la secuencia de salida resultante de todos 0 será interpretada por el receptor como procedentes de la entrada $I_0(\mathrm{x}) = 0$. Por lo tanto un número finito de los errores del canal, ha causado un número infinito de errores de decodificación. Dicho código se llama un

código catastrófico. Hay una manera fácil de evitar que un código convolutivo de razón $1/n$ sea catastrófico, es decir, al exigir que

$$\text{mcd}(g_1(x),\, g_2(x),\, \ldots\, ,\, g_n(x)) = 1.$$

Es bien sabido que esto implica que existen polinomios $a_i(x)$ tales que

$$\sum_{i=1}^{n} a_i(x) g_i(x) = 1.$$

A partir esto encontramos

$$\sum_{i=1}^{n} a_i(x^n) V_i(x^n) = \sum_{i=1}^{n} a_i(x^n) g_i(x^n) I_0(x^n) = I(x),$$

es decir, la entrada se puede determinar a partir de la salida y, además, un número finito de errores en $T(x)$ no puede causar un número infinito de errores de decodificación.

Hay dos maneras de describir la generalización de los códigos convolutivos de razón k/n. Ahora tenemos k registradores de cambio con secuencias de entrada $I_0(x)$, . . . , $I_k(x)$. Hay n secuencias de salida $V_i(x)$ $(i = 1,\ \ldots\ ,\ n)$, donde $V_i(x)$ se forma usando todos los registradores de cambio. En primer lugar, utilizando el método que se usó anteriormente. Ahora necesitamos kn polinomios $g_{ij}(x)$ $(i = 1,\ \ldots\ ,\ k\ ;\ j = 1,\ \ldots\ ,\ n)$ para describir la situación. Tenemos

$$V_j(x) = \sum_{i=1}^{k} g_{ij}(x) I_i(x).$$

Ya no es posible describir la codificación con un solo polinomio generador.

La siguiente manera de describir los códigos convolutivos de razón k/n que los convierte en códigos de bloque sobre un campo convenientemente elegido.

Sea F el campo cociente de $\mathbb{F}_2[x]$. Es decir, el campo de todas las series de Laurent de la forma

$$\sum_{i=r}^{\infty} a_i x^i, \qquad (r \in \mathbb{Z},\ a_i \in \mathbb{F}_2).$$

Consideramos k bits entrando a los diferentes registradores de cambio en el tiempo$t$ como un vector en $\mathbb{F}_2^k$. Esto significa que las secuencias de entrada se interpretan como vectores en F^k. Ahora consideramos los kn polinomios $g_{ij}(x)$ como los elementos de una matriz generadora G. Por supuesto, las n secuencias de salida pueden ser vistas como elementos de F^n. Esto nos lleva a la siguiente definición.

Definición. Un código convolutivo C de razón k/n es un subespacio de F^n dimensión k que tiene una base de k vectores de $\mathbb{F}[x]^n$. Estos vectores de la base son las filas de G.

Aunque esto muestra cierta analogía con los códigos de bloque estamos ocultando las dificultades mediante el uso de F y, además, la restricción sobre los elementos de G es muy severa. En la práctica todos los mensajes son finitos y podemos suponer que hay un silencio para $t < 0$. Esto significa que realmente no se necesita F y se puede hacer todo con $\mathbb{F}[x]$. Dado que esto no es un campo, habrá otras dificultades para este enfoque.

5.3. Diagrama de estado

Una manera eficaz de describir la acción de este codificador está dada por el diagrama de estado. Conectamos dos estados mediante una flecha solida si el registrador va de un estado a otro con un input de 0. Similarmente una flecha punteada si corresponde a un input de 1. Junto a estas flechas se indican las salidas de V_1 y V_2. Un secuencia de entrada I_0 corresponde a un recorrido a través del diagrama de estado.

Ejemplos.

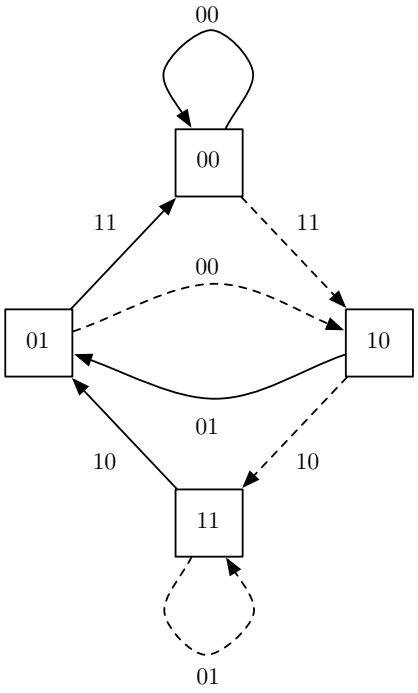


Diagrama de estado para el código convolutivo (2, 1, 3) de la Figura 1.

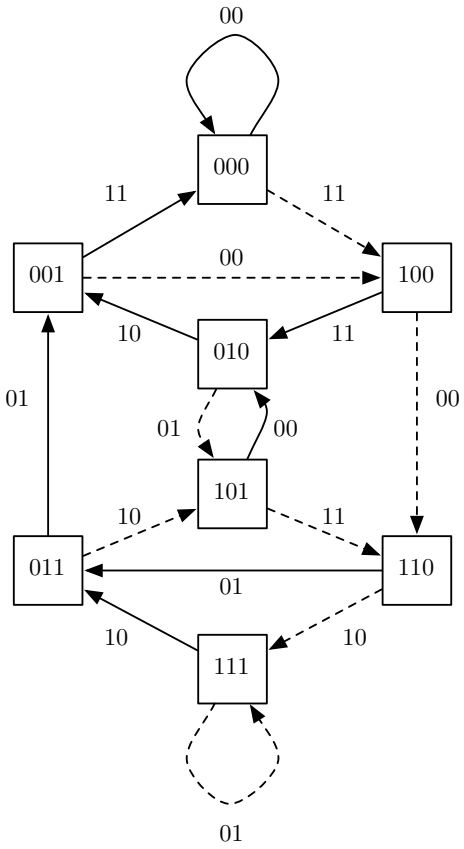


Diagrama de estado para el código convolutivo (2, 1, 4) de la Figura 2.

Sabemos que uno de los números más importantes vinculados a un código de bloque C es la distancia mínima. Para los códigos convolutivos existe un concepto similar que a su vez juega un papel central en el estudio de la decodificación. Este número se llama la *distancia libre* del código y se define como el peso mínimo de todas las secuencias de salida distintas de cero. En el ejemplo de la Figura 2, esta distancia libre es 6.

5.4. Diagrama de Trellis

Los diagramas de Trellis son algo turbios, pero en general son preferibles a los diagramas de estado, ya que representan la secuencia de tiempo lineal de los acontecimientos. El eje X es tiempo discreto y todos los estados posibles se muestran en el eje Y. Nos movemos horizontalmente a través del diagrama con el paso del tiempo. Cada transición significa que nuevos bits han llegado.

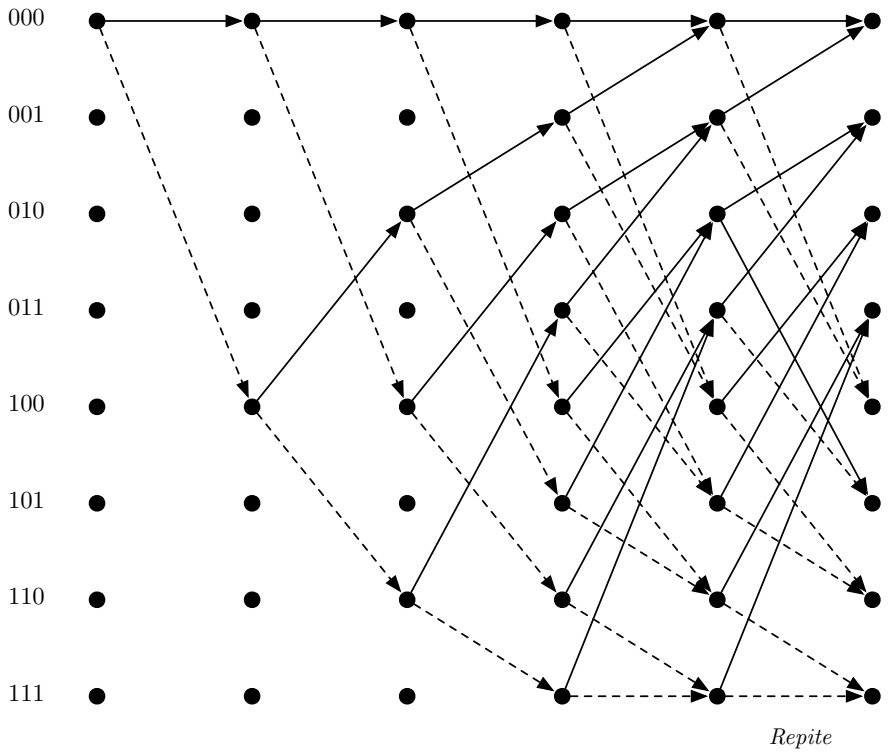


Diagrama de Trellis del código de la Figura 2.

El diagrama de Trellis se dibuja, alineando todos los estados posibles en el eje vertical. Entonces conectamos cada estado al siguiente estado, sólo hay dos opciones posibles en cada estado. Estos están determinados por la llegada de cualquiera un 0 o un 1 bit. Las flechas que están subiendo representan un bit 0 y las que bajan representan un bit 1. El diagrama de Trellis es único para cada código, lo mismo que el diagrama de estado. Podemos dibujar el diagrama por períodos hasta que queramos. Después de un período se repiten las transiciones posibles.

Este diagrama nos da una idea acerca de la decodificación: si una secuencia recibida no se ajusta a este gráfico, entonces fue recibido con errores, y tenemos que elegir la secuencia más cercana correcta. Los algoritmos de decodificación explotan esta idea.

5.5. Decodificación de Viterbi

"Decodificadores que implementan el algoritmo de Viterbi (AV) se utilizan actualmente en cerca de un billón de teléfonos celulares, que es probablemente el mayor número en cualquier aplicación. Sin embargo, el mayor consumidor actual de ciclos de procesador del AV es, probablemente, la radiodifusión de vídeo digital. Una estimación reciente de Qualcomm es que aproximadamente 10^{15} bits por segundo están siendo decodificados por el AV en televisión digital alrededor del mundo, cada segundo de cada día." de *"El algoritmo de Viterbi: Una historia personal"* de G. David Forney, Jr., presentado en la Conferencia de Viterbi en la Universidad del Sur de California, Los Angeles, 8 de marzo de 2005.

Hay varios algoritmos utilizados en la práctica para decodificar los códigos convolutivos. Son más o menos similares y todos matemáticamente no muy profundos. De hecho, se asemejan a la decodificación de códigos bloque que sólo compara la palabra recibida con todas las palabras. La decodificación de Viterbi es la aplicación más conocida de la decodificación probabilidad máxima. Aquí se limitan las opciones de forma sistemática en cada instante del tiempo.

Un decodificador de Viterbi examina toda una secuencia recibida de una longitud dada. El decodificador calcula la distancia para cada trayectoria en el diagrama de Trellis y toma una decisión basada en esta distancia. Todas trayectorias se siguen hasta que dos caminos convergen en un nodo. Entonces la trayectoria con la distancia menor se mantiene y el que

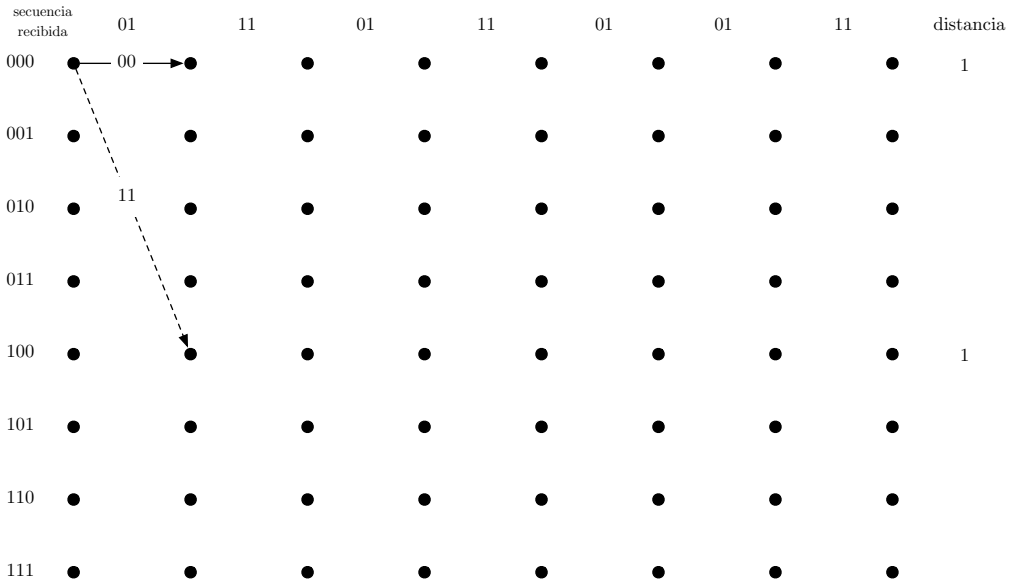
tiene distancia mayor se descarta, la métrica más utilizada es la distancia de Hamming. Las trayectorias seleccionadas se les llama sobrevivientes.

Para una secuencia de N bits, el número total de posibles secuencias recibidas es 2^N. De estas sólo 2^{kL} son válidos. El algoritmo de Viterbi aplica los principios de probabilidad máxima para limitar la comparación a 2^{kL} trayectorias sobrevivientes en lugar de comprobar todas las trayectorias posibles.

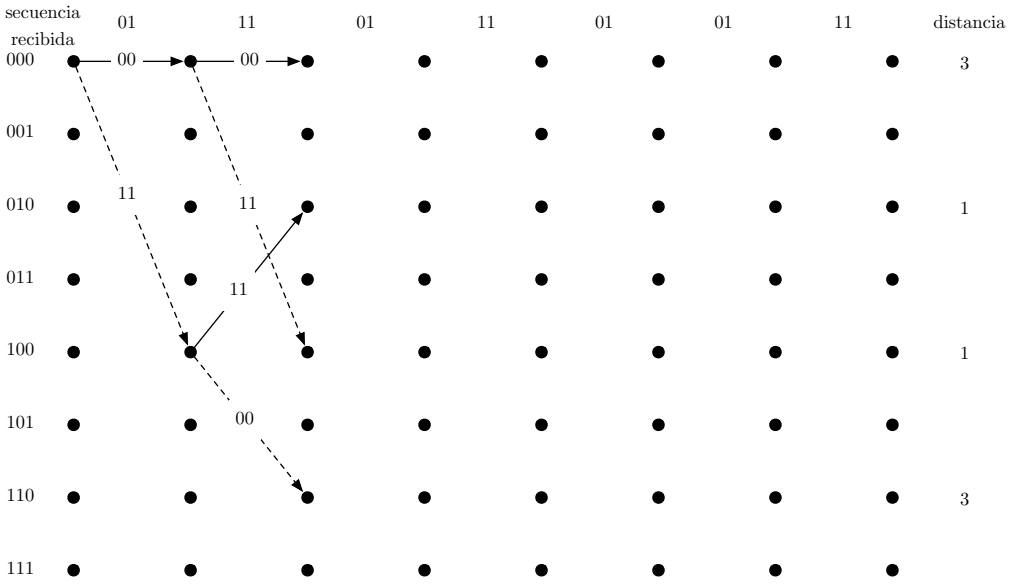
Hagamos un ejemplo para el código (2, 1, 4) de la Figura 2. Suponga que secuencia 1011000 fue enviada. Si no se han producido errores, se recibiría: 11 11 01 11 01 01 11 pero digamos que hemos recibido en su lugar: 01 11 01 11 01 01 11. Un error ha ocurrido. El primer bit se ha recibido como 0.

Vamos a decodificar la secuencia recibida 01 11 01 11 01 01 11 usando la decodificación de Viterbi.

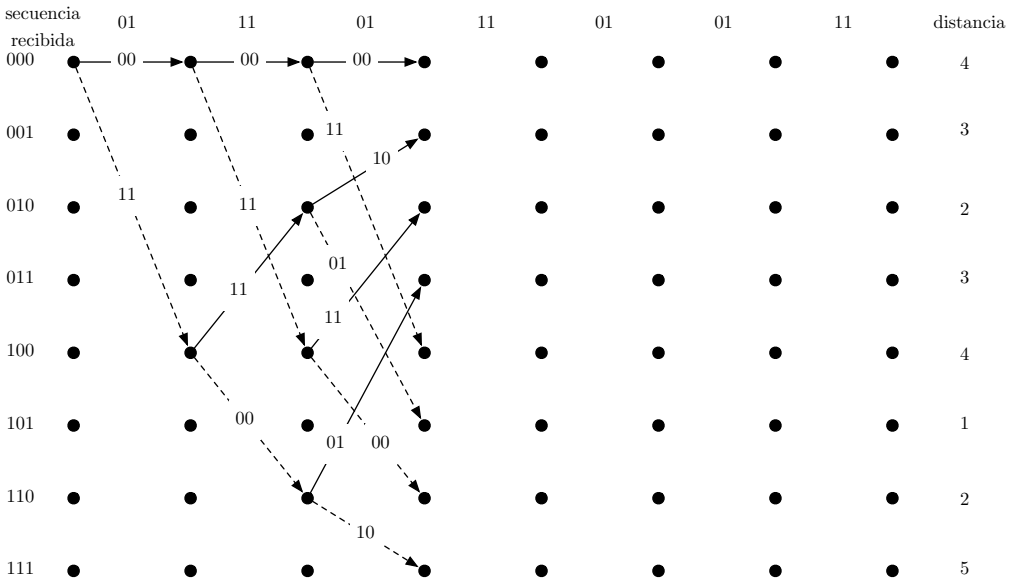
1. En $t = 0$, hemos recibido el bit 01. El decodificador inicia siempre en el estado 000. Desde este punto se tienen dos rutas disponibles, pero ninguna coincide con los bits de entrada. El decodificador calcula la distancia para ambas, y continua simultáneamente a lo largo de estas dos ramas. La distancia de Hamming de ambas ramas es igual a 1, lo que significa que uno de los dos bits fue igual a los bits de entrada.



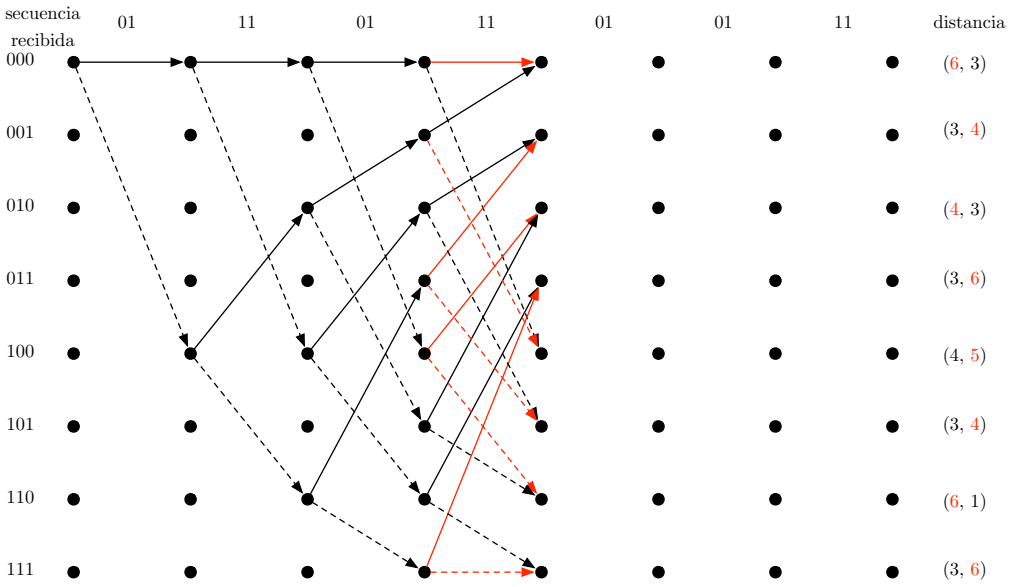
2. En $t = 1$, recibimos 11 y el decodificador se despliega de estos dos estados posibles a cuatro estados. Calculamos las distancias de Hamming para 11 con las cuatro posibles salidas y les agregamos respectivamente las distancias previamente calculadas.



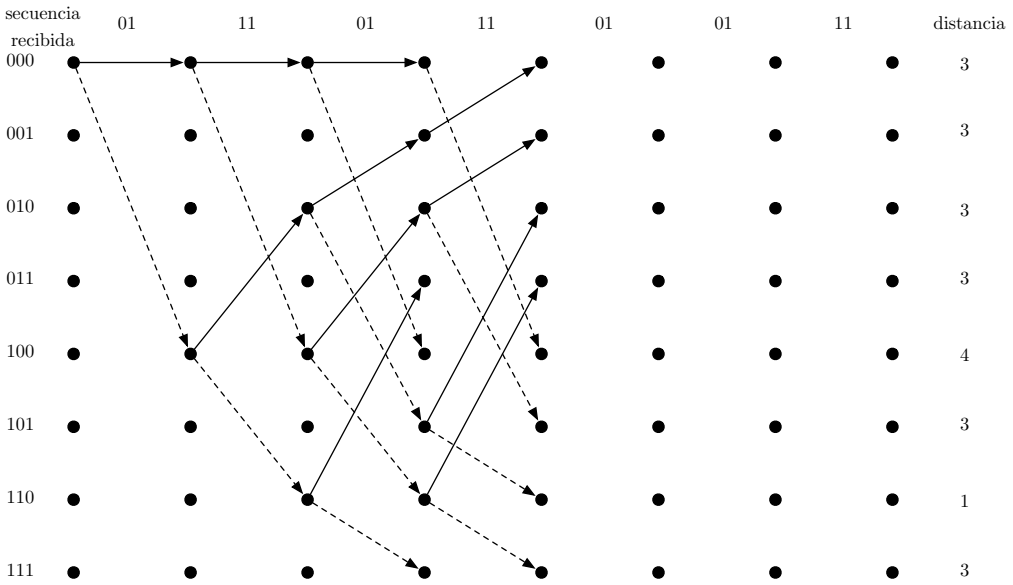
3. En $t = 2$, los cuatro estados se expanden a ocho para mostrar todos los caminos posibles. Se calculan las diastancias de Hamming para 01 y se agregan a las distancias previamente calculadas de $t = 1$.



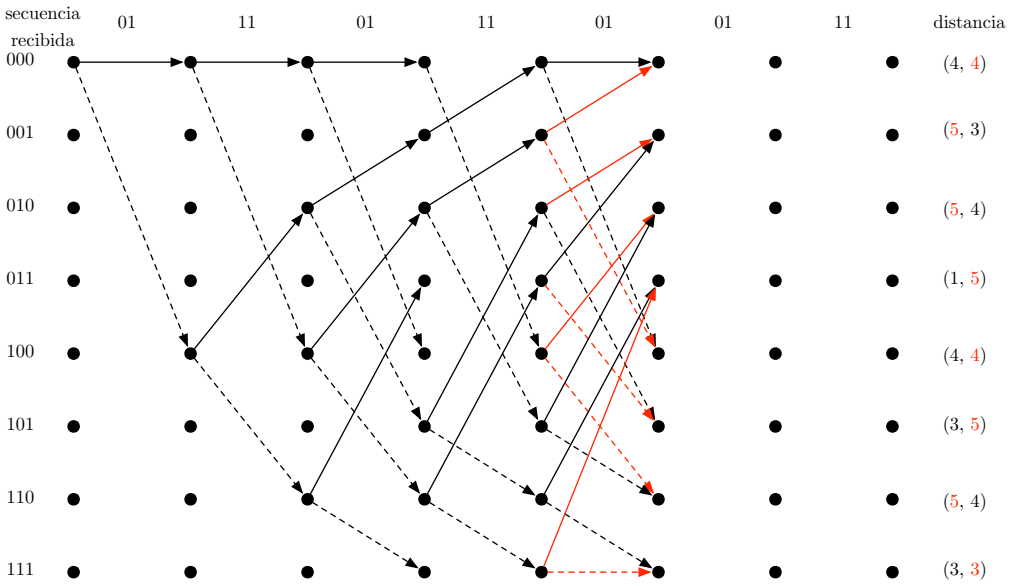
4. En $t = 3$, las trayectorias avanzan hacia adelante y ahora dos trayectorias convergen en un nodo. Por el principio de máxima probabilidad, en cada nodo se descarta la ruta con la distancia más grande debido a que es menos probable. Al descartar trayectorias en cada nodo ayuda a reducir el número de trayectorias que tienen que ser examinados y eso da al método de Viterbi su fuerza. En el diagrama las trayectorias en rojo se descartan.



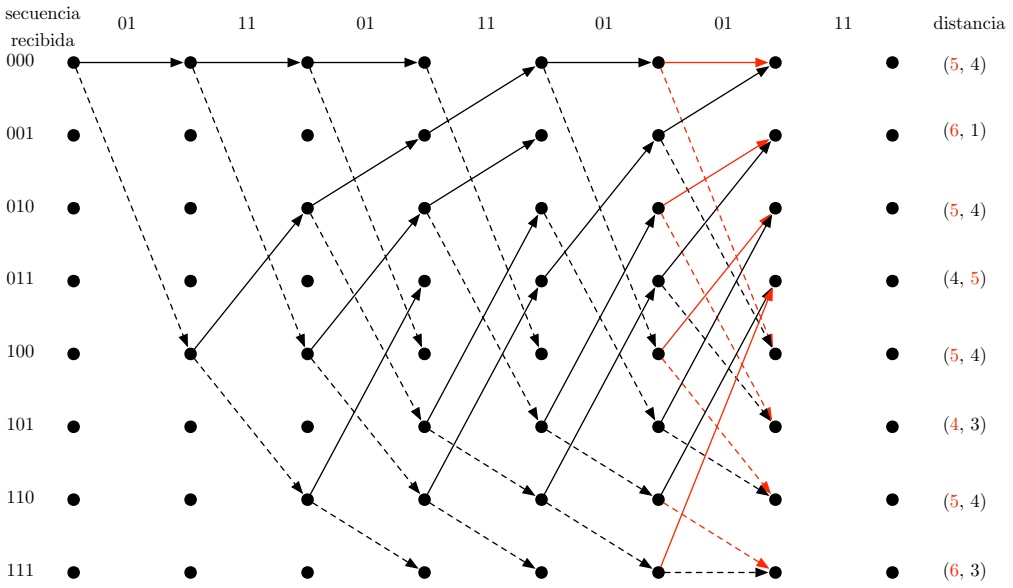
5. Después de descartar las trayectorias con la distancia más grande, nos quedan las siguientes trayectorias.



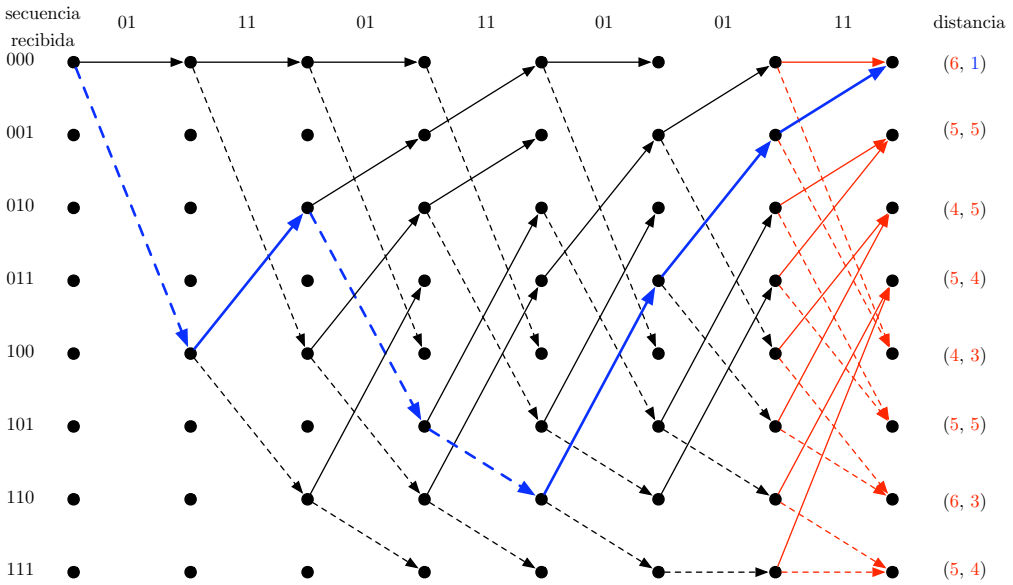
6. Volvemos a ir hacia adelante y calcular nuevas métricas. En el siguiente nodo, una vez más la convergencia de trayectorias y otra vez que descartar aquellos con menor métrica. Si las dos distancias acumuladas son iguales, algunas personas utilizan un lanzamiento de moneda para elegir el cual trayectoria sobrevive. Otros simplemente eligen una de ellas consistentemente, es decir, la rama superior o de la rama inferior (en nuestro caso elegimos la rama superior). Probablemente no importa el método que use.



7. Continuamos avanzando hacia adelante y calculando nuevas métricas y descartando aquellas trayectorias con distancia mas grande.



8. En este paso el diagrama de Trellis se ha completado. Ahora elegimos la trayectoria con la distancia más pequeña y tenemos un ganador. El camino azul trazado por los estados 000, 100, 010, 101, 110, 011, 001, 000 y correspondiente a los bits 1011000 es la secuencia decodificada.



La decodificación de Viterbi tiene la ventaja de que tiene un tiempo de decodificación fijo. Es muy adecuado para la implementación de hardware. Sin embargo, sus requisitos computacionales crecen exponencialmente en función de la restricción de longitud, por lo que suele ser limitada en la práctica a las restricciones de longitud de $K = 9$ o menos. *Stanford Telecom* produce un decodificador de Viterbi con $K = 9$, que opera a velocidades de hasta 96 kbps, y uno con $K = 7$ que opera a velocidades de hasta 45 Mbps. *Advanced Wireless Technologies* ofrece un decodificador de Viterbi con $K = 9$, que opera a velocidades de hasta 2 Mbps. La Ley de Moore se aplica a los decodificadores de Viterbi, así como a los microprocesadores, por lo que debería considerar las tasas mencionadas anteriormente como una fotografía instantánea de su estado a principios de 1999.

Durante años, la codificación convolucional con decodificación de Viterbi ha sido la técnica predominante utilizada en las comunicaciones espaciales.

5.6. Códigos convolutivos con *Mathematica*

En esta sección programaremos el código convolutivo (2, 1, 4) de la Figura 2. Por simplicidad a los estados los enumeraremos del 1 al 8 según su representación en base 2 mas uno por ejemplo $6 \rightarrow [1, 0, 1]$.

Construcción del código:

Para codificar cualquier sucesión construimos la siguiente función que también nos devuelve el estado final del codificador ya que puede ser útil mas adelante.

```
conv = Function[s = {0, 0, 0}; i = #1; m = Length[i];
   h[1] = s[[3]]; h[2] = s[[2]]; h[3] = s[[1]];
   For[j = 1, j <= m, j++, h[j + 3] = i[[j]]];
   H = Array[h, 2 m];
   For[k = 1, k <= m, k++,
    out1[(1 + 2 (k - 1))] =
     Mod[H[[k + 3]] + H[[k + 2]] + H[[k + 1]] + H[[k]], 2];
    out1[2 k] = Mod[H[[k + 3]] + H[[k + 2]] + H[[k]], 2]];
   OUT = Array[out1, 2 m];
   sf = {h[m + 3], h[m + 2], h[m + 1]}.{4, 2, 1} + 1; {OUT, sf}];
```

Decodificación:

Para programar la descodificación de Viterbi primero vamos a construir una tabla que contiene la información del diagrama de estado.
Como a cada estado se puede llegar de dos formas diferentes construiremos una función que nos diga de que estados posibles proviene cada uno y para eso primero construimos las siguiente funciones auxiliares:

```
q2 = Function[m = #1; l = Mod[m, 2]; If[l === 1, n = (m + 1)/2, n =
m/2]; n];
```

```
iq2 = Function[x = #1; k1 = 1; While[q2[k1] != x, k1++]; {k1, k1 +
1}];
```

```
aux = Function[x = #1; r = Mod[x - 1, 8] + 1; q = Quotient[x - 1, 8];
{r, q}];
```

utilizando las funciones anteriores construimos la siguiente función que nos proporciona la información de las dos posibles formas de llegar a un estado por ejemplo si escribimos std[6] nos devuelve {{3, 1}, {4, 1}} esto quiere decir que al estado 6 podemos llegar partiendo del estado 3 y avanzando con un 1 o estando en el estado 4 y avanzando con también con 1.

```
std = Function[x1 = #1; b1 = iq2[x1]; c1 = aux[b1[[1]]];
   d1 = aux[b1[[2]]]; {c1, d1}];
```

Ahora lo que queremos es tener la salida que deja cada movimiento que se realiza en el diagrama de estado por ejemplo si estamos en el estado 3 y avanzamos con un 1 la salida seria 01, para eso escribimos las siguientes unciones:

```
ip = Tuples[{0, 1}, 4];
```

```
cd = Function[x2 = #1; s1 = x2[[1]]; e1 = x2[[2]]; h1 = ip[[s1 + 8
e1]];
   a1 = Mod[h1.{1, 1, 1, 1}, 2]; b2 = Mod[h1.{1, 1, 0, 1}, 2]; {a1,
b2}];
```

y la siguiente función es la que queríamos ya que nos da información de las dos posibles formas de llegar a un estado y su correspondiente salida por ejemplo si escribimos bst[6] nos devuelve {3, {0, 1}, 4, {1, 0}} esto quiere decir que al estado 6 podemos llegar del estado 3 con salida 01 o del estado 4 con una salida de 10.

```
bst = Function[x3 = #1;
   a2 = std[x3]; {a2[[1, 1]], cd[a2[[1]]], a2[[2, 1]], cd[a2[[2]]]}];
```

Ahora y sólo por ahorrar cálculos escribimos las salidas del 1 al 8 de la función bst.

```
bk = Array[bst, 8];
```

Entonces ya teniendo la tabla anterior ya lo único que nos falta para programar la decodificación de Viterbi es una función que calcule la distancia de Hamming

```
hdist = Function[a = #1; b = #2; Total[Mod[a + b, 2]]];
```

Y ahora escribimos la función que va a descartar uno de los dos posibles caminos a cada estado:

```
vtb = Function[sta = bk[[#1]]; o = #2; d = #3;
   h2[1] = {hdist[sta[[2]], o] + d[[sta[[1]]]], sta[[1]]};
   h2[2] = {hdist[sta[[4]], o] + d[[sta[[3]]]], sta[[3]]};
   H1 = Sort[{h2[1], h2[2]}, #1[[1]] < #2[[1]] &]; H1[[1]]];
```

Recordemos que la función vtb no puede trabajar sino hasta el cuarto paso en el diagrama de trellis entonces escribimos todos los posibles caminos y salidas asta el tercer paso.

```
v3 = {{{0, 0, 0, 0, 0, 0}, {1, 1, 1}}, {{1, 1, 1, 1, 1, 0}, {5, 3,
2}}, {{0,
     0, 1, 1, 1, 1}, {1, 5, 3}}, {{1, 1, 0, 0, 0, 1}, {5, 7, 4}},
{{0, 0, 0,
     0, 1, 1}, {1, 1, 5}}, {{1, 1, 1, 1, 0, 1}, {5, 3, 6}}, {{0, 0,
1, 1, 0,
     0}, {1, 5, 7}}, {{1, 1, 0, 0, 1, 0}, {5, 7, 8}}};


For[i4 = 1, i4 < 9, i4++, v[3][i4] = v3[[i4, 2]]];
```

Las siguientes funciones son muy parecidas la primera función nos devuelve las trayectorias posibles en el diagrama de trellis hasta determinado tiempo, la segunda nos da la trayectoria con distancia mínima.

```
trellis = Function[sig = #1; m1 = #2;
   For[i2 = 1, i2 <= 8, i2++,
    dn[3][i2] = hdist[v3[[i2, 1]], Take[sig, {1, 6}]]];
   For[j1 = 4, j1 <= m1, j1++, Dn[j1 - 1] = Array[dn[j1 - 1], 8];
    V[j1 - 1] = Array[v[j1 - 1], 8];
    For[i3 = 1, i3 <= 8, i3++,
     s2 = vtb[i3, Take[sig, {2 j1 - 1, 2 j1}], Dn[j1 - 1]];
     dn[j1][i3] = s2[[1]];
     v[j1][i3] = Join[V[j1 - 1][[s2[[2]]]], {i3}]]]; V[m1] =
Array[v[m1], 8]];

dec = Function[sig = #1; m1 = Length[sig]/2;
   For[i2 = 1, i2 <= 8, i2++,
    dn[3][i2] = hdist[v3[[i2, 1]], Take[sig, {1, 6}]]];
   For[j1 = 4, j1 <= m1, j1++, Dn[j1 - 1] = Array[dn[j1 - 1], 8];
    V[j1 - 1] = Array[v[j1 - 1], 8];
```

```
    For[i3 = 1, i3 <= 8, i3++,
     s2 = vtb[i3, Take[sig, {2 j1 - 1, 2 j1}], Dn[j1 - 1]];
     dn[j1][i3] = s2[[1]];
     v[j1][i3] = Join[V[j1 - 1][[s2[[2]]]], {i3}]]; V[m1] =
Array[v[m1], 8];
   Dn[m1] = Array[dn[m1], 8]; p = Position[Dn[m1], Min[Dn[m1]]][[1,
1]];
   V[m1][[p]]];
```

La función que sigue transforma una trayectoria en su correspondiente entrada de unos y ceros.

```
unoce = Function[sn = #1; s3 = {}; t3 = Tuples[{0, 1}, 3]; Do[s3 =
Join[s3, {t3[[i5, 1]]}], {i5, sn}];
    s3];
```

```
deco = Function[unoce[dec[#1]]];
```

Por último la función deco escrita anteriormente es una composición de funciones nos da la decodificación completa.

Conclusiones

Este trabajo debe su origen a la labor pionera de Claude Shannon en 1948 en lograr una comunicación fiable sobre un canal de transmisión con ruido. Claude Shannon arrojó el guante a las futuras generaciones de investigadores. En aquellos días predigitales, los canales de comunicación tales como las líneas telefónicas o bandas de radio eran particularmente susceptibles a las interrupciones eléctricas o electromagnéticas conocido como ruido. Shannon demostró el resultado contrario a la intuición de que no importa lo ruidoso de un canal, la información podría ser enviada a través del ruido libre de errores. Todo lo que necesitaba era una manera de agregar redundancia suficiente a la información para que los errores pueden ser corregidos. También demostró que había un límite en el grado de eficiencia de los códigos correctores de errores (la cantidad mínima de información adicional que garantice cerca de cero errores). Puesto que códigos más largos toman más tiempo para enviarse un código de longitud mínima implicaba una velocidad máxima de transmisión (*el límite de Shannon*). Por último, Shannon demostró que los códigos que se acercan a ese límite deben existir, pero no mostró cómo encontrarlos. El diseño de códigos buenos y de métodos eficaces de decodificación, iniciado por Hamming y otros en la década de 1950, ha ocupado las energías de muchos investigadores desde entonces.

Los códigos vistos en este trabajo fueron importantes en su época y hasta la fecha algunos de ellos siguen siendo utilizados en la practica pero están empezando a ser remplazados por códigos mejores en el sentido que se aproximan más al limite de Shannon, por ejemplo los *Turbo Codes* y los *LDPC* (*Low Density Parity Check Codes*).

Bibliografía

Richard E. Klima, Neil Sigmon, Ernest Stitzinger. *Applications of Abstract Algebra with Maple* . CRC Press; 1 edición, 1999.

J. H. van Lint. *Introduction to Coding Theory*. Springer; 3 edición, 1998.

Daniel J. Costello Jr., Joachim Hagenauer, Hideki Imai, Stephen B. Wicker. *Applications of Error-Control Coding.* IEEE Transactions on Information Theory, Vol. 44, No. 6, 1998.

Ben Cooke. *Reed-Muller Error Correcting Codes*. MIT Undergraduate Journal of Mathematics Vol. 1, MIT Department of Mathematics, 1999.

G. David Forney Jr. *The Viterbi Algorithm: A Personal History.* Presentado en la Conferencia de Viterbi, University of Southern California, Los Angeles, Marzo 8, 2005.

Chip Fleming. *A tutorial on Convolutional Coding with Viterbi Decoding*. Spectrum Applications, 2002.

Fdo. Cándido Piñeiro Gómez. *Una Aplicación reciente de la Matemática: La Teoría de Códigos.* http://www.uhu.es/candido.pineiro/historia/codigopren.pdf

Charan Langton. *Coding and decoding with Convolutional Codes.* http://www.complextoreal.com/chapters/convo.pdf

Páginas web consultadas:

http://en.wikipedia.org/wiki/Error_detection_and_correction

http://en.wikipedia.org/wiki/Hamming_code

http://en.wikipedia.org/wiki/Reed–Muller_code

http://en.wikipedia.org/wiki/Reed–Solomon_error_correction

http://en.wikipedia.org/wiki/Convolutional_code