CÓDIGOS CÍCLICOS LRC-LCD

YISETH KARINA RODRÍGUEZ CÁCERES

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE CIENCIAS
ESCUELA DE MATEMÁTICAS
BUCARAMANGA
2023

CÓDIGOS CÍCLICOS LRC-LCD

YISETH KARINA RODRÍGUEZ CÁCERES

Trabajo de grado para optar al título de
Matemática

Director
Wilson Olaya León
Profesor
Escuela de Matemáticas

Codirectora
Diana Haidive Bueno Carreño
Profesora
Departamento de Ciencias Naturales y Matemáticas
Pontificia Universidad Javeriana
Cali

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE CIENCIAS
ESCUELA DE MATEMÁTICAS
BUCARAMANGA
2023

DEDICATORIA

A mis padres y hermanos,

Por su amor incondicional y apoyo constante, este logro es suyo tanto como mío. Gracias por ser mi inspiración y motivación en este proyecto.

AGRADECIMIENTOS

En primer lugar, agradezco a Dios, mi guía constante, por llenarme de fortaleza y acompañarme en cada etapa de mi vida.

A mis queridos padres, Sonia Cáceres y Luis Alfredo Rodríguez, les dedico un sincero agradecimiento. Su ejemplaridad como padres se ha reflejado en los sacrificios que han hecho para brindarme las herramientas necesarias para alcanzar mis sueños. Han respaldado incondicionalmente mis metas, siendo mi apoyo constante frente a cualquier adversidad. Con su cariño, han sido el impulso que necesitaba para perseguir mis objetivos. También agradezco a mis familiares, especialmente a Lilibeth, Julián, Deisy, Daira y Felipe, por su valiosa contribución a este logro.

De manera especial, a mi profesor y director de tesis, Wilson Olaya, por haberme guiado no solo en la elaboración de este trabajo, sino a lo largo de mi carrera, brindandome el apoyo necesario para mi crecimiento profesional y personal. Asimismo, agradezco a mi codirectora, Diana Bueno, por dedicar su tiempo y sabiduría para ayudarme a comprender los resultados presentados en esta tesis. Me enorgullece contar con su apoyo y expreso mi total admiración.

Además, quiero expresar mi agradecimiento a todos mis compañeros, muchos de los cuales se han convertido en verdaderos amigos, su apoyo y amistad han sido un pilar esencial en mi experiencia universitaria. Un agradecimiento especial a César David, quien desde el primer semestre ha sido un apoyo constante y una fuente de motivación.

Por último, agradezco a la Universidad Industrial de Santander, que me ha exigido tanto, pero al mismo tiempo me ha permitido obtener mi tan anhelado título.

Con profunda gratitud,

Yiseth Karina Rodríguez Cáceres.

CONTENIDO

pág.

LISTA DE CUADROS

pág.

LISTA DE FIGURAS

pág.

RESUMEN

TÍTULO: CÓDIGOS CÍCLICOS LRC-LCD [*]

AUTOR: YISETH KARINA RODRÍGUEZ CÁCERES [**]

PALABRAS CLAVE: CÓDIGOS LINEALES, CÓDIGOS CÍCLICOS, CÓDIGOS CORRECTORES DE ERRORES, CÓDIGOS LINEALES LRC, CÓDIGOS LINEALES LCD, ALMACENAMIENTO DISTRIBUIDO, LOCALIDAD.

DESCRIPCIÓN:

En el contexto actual, donde la teoría de la información y los medios digitales están en constante evolución, se enfrentan desafíos cruciales, como garantizar la integridad y confidencialidad de los datos sensibles, así como gestionar eficientemente grandes volúmenes de información. La aplicación de códigos correctores de errores emerge como una herramienta esencial para abordar estos desafíos. Este trabajo se enfoca en los códigos cíclicos localmente recuperables (LRC) y códigos cíclicos duales complementarios (LCD), presentando una combinación estratégica de ambos. Estos códigos no solo corrigen errores en la transmisión de datos, sino que también desempeñan un papel crucial en la protección de datos sensibles, utilizando técnicas como el enmascaramiento de datos. Además, se exploran aplicaciones prácticas en almacenamiento distribuido y se destaca la implementación en SageMath para la construcción y análisis de propiedades específicas de los códigos lineales.

En específico, esta investigación se centra en códigos cíclicos que posean propiedades tanto de ser códigos lineales localmente recuperables (LRC) como códigos lineales duales complementarios (LCD).

---

[*] Trabajo de grado

[**] Facultad de Ciencias. Escuela de Matemáticas. Director: Wilson Olaya León, Doctor en Ciencias Matemáticas. Codirectora: Diana Haidive Bueno, Doctora en Matemáticas.

ABSTRACT

TITLE: LRC-LCD CYCLIC CODES [*]

AUTHOR: YISETH KARINA RODRÍGUEZ CÁCERES [**]

KEYWORDS: LINEAR CODES, CYCLIC CODES, ERROR CORRECTING CODES, LINEAR LRC CODES, LINEAR LCD CODES, DISTRIBUTED STORAGE, LOCALITY.

DESCRIPTION:

In the current context, where information theory and digital media are constantly evolving, crucial challenges are faced, such as ensuring the integrity and confidentiality of sensitive data, as well as efficiently managing large volumes of information. The application of error-correcting codes emerges as an essential tool to address these challenges. This work focuses on cyclic locally recoverable codes (LRC) and dual-complementary cyclic codes (LCD), presenting a strategic combination of both. These codes not only correct errors in data transmission but also play a crucial role in the protection of sensitive data, using techniques such as data masking. Furthermore, practical applications in distributed storage are explored, and the implementation in SageMath is highlighted for the construction and analysis of specific properties of linear codes.

Specifically, this research centers on cyclic codes that possess properties of being both locally recoverable linear codes (LRC) and dual-complementary linear codes (LCD).

---

[*] Undergraduate Thesis

[**] Faculty of Sciences. School of Mathematics. Director: Wilson Olaya León, Doctor in Mathematical Sciences. Co-Director: Diana Haidive Bueno, Doctor in Mathematics.

INTRODUCCIÓN

Actualmente vivimos en la era de la información, por lo cual garantizar la confiabilidad y privacidad de los datos es uno de los retos más importantes. En vista de que cuando se transmite información por medios analógicos existe la probabilidad de que se cometa algún error debido a la presencia de ruido o interferencia en el medio, lo deseado es que cuando se transmite un mensaje a través de un canal, el receptor reciba el mismo mensaje que fue enviado por el emisor, o en caso contrario que pueda determinar si hubo alguna alteración en la información y, si es posible, llegar a corregirlo. Es aquí donde entra la teoría de códigos y lo que se conoce como códigos correctores de errores, cuya estrategia consiste en incluir en el mensaje información de modo que el receptor pueda detectar si hubo errores en la transmisión, corregir un número limitado de errores y así obtener el mensaje que le fue enviado por el emisor. Los códigos correctores de errores fueron introducidos a mediados del siglo pasado, y la familia más representativa por su estructura algebraica es la familia de los códigos cíclicos, estos son códigos lineales (subespacios vectoriales sobre un cuerpo finito) en el que cada cambio cíclico de una palabra es de nuevo una palabra del código. Sin embargo, este no es el único problema que se presenta en la transmisión de información digital, si bien estos códigos fueron diseñados inicialmente con el objetivo de detectar y corregir errores en la transmisión de datos, la sociedad de la información presenta otra serie de problemas cuya solución se ha encontrado en el uso de códigos correctores de errores, como por ejemplo el almacenamiento distribuido (utilizados para el almacenamiento en la nube), privacidad de datos sensibles (utilizados para enmascaramiento de datos) y criptografía postcuántica (protocolo criptográfico McElice).

Existe una amplia cantidad de empresas tecnológicas y de medios de comunicación que requieren almacenar una gran cantidad de datos de tal manera que exista algún mecanismo que permita evitar la pérdida de información. Una solución para esto, sería contar con copias de seguridad de paquetes de datos en diferentes nodos, dicho proceso se conoce como replicación. Sin embargo, para grandes cantidades de datos, dicho proceso no es rentable, por lo cual se utilizan tipos de códigos que permitan una mayor capacidad de almacenamiento, tal como lo son los códigos Reed-Solomon (RS). Un ejemplo de medios sociales que emplean los códigos de Reed-Solomon en sus sistemas de almacenamiento distribuido son Facebook y Google. En particular, cuando ocurre un fallo de nodo

que almacena ciertas cantidades de datos, es necesario poder reconstruir de manera rápida dicha información de modo que siempre se encuentre disponible para los usuarios.

Los códigos localmente recuperables (LRC por sus iniciales en inglés, locally recoverable codes) se introducen motivados por el uso de la teoría de códigos aplicada a los sistemas de almacenamiento distribuido o almacenamiento en la nube, estos códigos permiten la recuperación rápida de información, aumentar la confiabilidad y eficacia de dichos sistemas, haciendo posible abordar el problema sobre fallos de nodo. Estos códigos permiten la reparación de nodo de almacenamiento fallido mediante el uso de un pequeño número de otros nodos en el sistema. En 2012 D. Papailiopoulos A. Dimakis [1] mostraron la existencia de un código LRC óptimo que logra velocidades de datos arbitrariamente altas, bajo la métrica de recuperación de la localidad, proporcionando un límite de información que relaciona la localidad, la distancia del código y la capacidad de almacenamiento de cada nodo.

Un código lineal q-ario C se denomina código lineal dual complementario (LCD por sus iniciales en inglés, linear complementary dual) si cumple que la intersección del código C con su código dual $C^{\perp}$ de como resultado $\{0\}$. Estos códigos fueron introducidos por Massey [2], en 1992. Los códigos LCD se han empleado en enmascaramiento de datos ya que mejoran la seguridad de información sensible, razón por la cual muchos académicos estudian las propiedades y construcciones de los códigos LCD. Particularmente, en [3] Yang y Massey, caracterizaron los códigos cíclicos LCD. Para mantener la fiabilidad del sistema y la seguridad contra ataques externos, Carlet y Guilley [4] construyeron códigos LCD cíclicos con una gran distancia mínima y gran velocidad, que se pueden usar contra la fuga y la corrupción de información. Para proteger

---

[1] DIMITRIS, Papailiopoulos y ALEXANDROS, Dimakis. “Locally Repairable Codes”. En: *IEEE Transactions on Information Theory* 60.10 (2014), págs. 5843-5855. DOI: 10.1109/TIT.2014.2325570.

[2] Massey JAMES. “Linear codes with complementary duals”. En: *Discrete Mathematics* 106-107 (1992), págs. 337-342. DOI: https://doi.org/10.1016/0012-365X(92)90563-U.

[3] YANG, Xiang y MASSEY, James. “The condition for a cyclic code to have a complementary dual”. En: *Discrete Mathematics* 126.1 (1994), págs. 391-393. DOI: https://doi.org/10.1016/0012-365X(94)90283-6.

[4] CLAUDE, Carlet y SYLVAIN, Guilley. “Complementary dual codes for counter-measures to side-channel attacks”. En: *Advances in Mathematics of Communications* 10.1 (2016), págs. 131-150. DOI: 10.3934/amc.2016.10.131.

información sensible el enmascaramiento es el método más estudiado, donde los datos sensibles son una palabra del código y la máscara es una palabra del código dual complementario. La idea es que las máscaras actúen como ruido agregado intencionalmente y que la información sensible sean palabras clave, para más información recomendamos [5].

Por lo tanto, los códigos que son LRC y LCD tienen mayor utilidad en los sistemas de almacenamiento de datos, ya que ofrecen soluciones para los dos problemas mencionados anteriormente (privacidad y almacenamiento distribuido) al mismo tiempo. Los códigos cíclicos LRC que también son códigos cíclicos LCD, se denominan códigos cíclicos LRC-LCD.

En el primer capítulo de este trabajo, se presentan los preliminares de códigos lineales, detallando propiedades importantes en la detección y corrección de errores.

En el segundo capítulo se encuentran los códigos cíclicos. Exploraremos diversas nociones esenciales, tales como las propiedades que se obtienen de su representación algebraica como ideales de un anillo cociente.

En el tercer capítulo se describen los códigos LCD (códigos complementarios duales) y su importacia para proteger información sensible a traves del enmascaramiento de datos. Se analiza su funcionamiento y se muestra más detalladamente una de las aplicaciones de gran relevancia de estos códigos. También se muestran resultados relevantes y se proporciona información necesaria para la creación de códigos que son LCD.

En el cuarto capítulo, se profundiza en las características y propiedades de los códigos LRC (códigos localmente recuperables). Se examina su rendimiento en diferentes escenarios. Además, se explican las ventajas que tiene la recuperación local para el almacenamiento distibuido o almacenamiento en la nube.

En el quinto capítulo, se aborda la construcción de códigos cíclicos LRC que también son LCD. Este enfoque se complementa con la presentación de un algoritmo diseñado para simular esta construcción, ofreciendo como resultado códigos cíclicos LRC-LCD.

---

[5] *et al.* BRINGER Julien. “Orthogonal Direct Sum Masking”. En: (jun. de 2014), págs. 40-56. DOI: 10.1007/978-3-662-43826-8_4.

A lo largo de este trabajo se muestran implementaciones en SageMath [6] para la visualización de algunos de los resultados y definiciones importantes presentados en cada capítulo.

[6] https://www.sagemath.org/

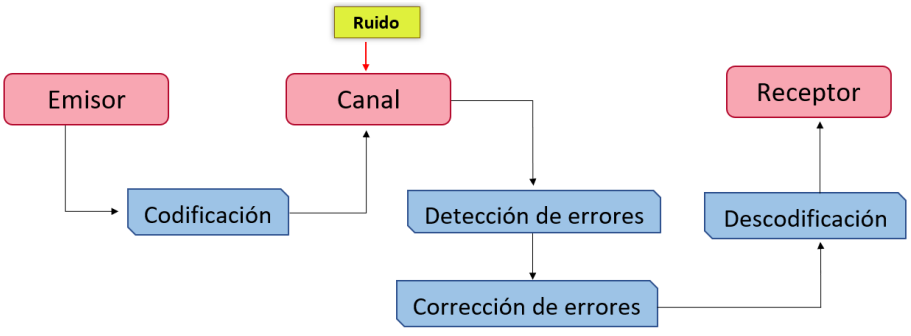
1. CÓDIGOS LINEALES

1.1. TRANSMISIÓN DE INFORMACIÓN

La era de la información presenta el desafío de garantizar la confiabilidad y privacidad de los datos cuando se transmiten por medios analógicos con presencia de ruido o interferencia en el canal. Esto se debe a que el objetivo es que el receptor reciba un mensaje igual al que fue enviado por el emisor y, en caso contrario, se busca poder detectar cualquier alteración en la información y dado el caso poder corregirlo. Es aquí donde la teoría de códigos desempeña un rol fundamental, particularmente los códigos correctores de errores juegan un papel crucial en la transmisión de información ya que estos códigos permiten la detección y corrección de errores en la transmisión de datos.

El siguiente esquema ilustra el proceso de transmisión de información mediante el uso de códigos correctores de errores para preservar la calidad de la información que es transmitidaa través de un canal, en el cual se pueden cometer errores debido a la presencia de ruido.

Figura 1.1: Esquema general de transmisión de información con detección y corrección de errores.



Fuente: Elaboración propia.

En el contexto de los códigos correctores de errores, utilizaremos $\mathbb{F}_q$ para referirnos al cuerpo finito con q elementos, donde $q = p^n$ con $n \in \mathbb{N}$ y p un número primo. Para denotar el conjunto de enteros positivos desde 1 hasta n, utilizaremos $[n]$ y nos enfocaremos en los códigos de bloque de longitud n sobre $\mathbb{F}_q$, es decir, $C \subseteq \mathbb{F}_q^n$.

A continuación, exploraremos los códigos lineales, los cuales son un tipo particular de códigos correctores de errores que han capturado considerablemente la atención en la teoría de la información y se han convertido en un foco principal de estudio debido a la estructura distintiva que presentan.

1.2. CÓDIGOS LINEALES

Los códigos lineales constituyen una clase de códigos correctores de errores, los cuales fueron introducidos por Shannon, Hamming y Golay en la década de 1940. Estos códigos son de gran importancia porque permiten que la transmisión de datos a través de canales de comunicación ruidosos sea más eficiente y confiable, además de permitir mayor velocidad de transmisión de datos y capacidad de almacenamiento. Estos códigos son los más estudiados e implementados ampliamente en la industria de la comunicación debido a que posee estructura de espacio vectorial.

A continuación se presentan conceptos básicos sobre los códigos lineales necesarios para la comprensión del presente trabajo.

Definición 1.2.1. *Un código lineal C es un subespacio vectorial de $\mathbb{F}_q^n$. Denotamos el código lineal de dimensión k y distancia mínima d como el $[n, k, d]$ código q-ario, donde la distancia mínima se define como el mínimo de las distancias de Hamming entre elementos diferentes del código (también llamados palabras código), es decir, $d = min\{d(u, v) : u, v \in C, u \neq v\}$, con $d(u, v) = \#\{i : u_i \neq v_i\}$, para $C \subset \mathbb{F}_q^n$.*

La distancia mínima d es un parámetro importante en un código lineal C, ya que determina su capacidad para detectar y corregir errores.

Teorema 1.2.2. *Sea C un $[n, k, d]$ código lineal, entonces C puede:*

1. *Detectar $d - 1$ errores.*
2. *Corregir hasta $\lfloor \frac{d-1}{2} \rfloor$ errores.*

Demostración. 1. Supongamos que se envía una palabra código x, y se recibe un vector y con, a lo más, $d - 1$ errores. En este caso, no es posible que y sea una palabra del código, ya que la distancia mínima en C es d, y tendríamos que la distancia $d(x, y) \leq d - 1$ es estrictamente menor que d. Por lo que podemos concluir que se han cometido errores en la transmisión.

2. Sea $t = \lfloor \frac{d-1}{2} \rfloor$, y supongamos que se envía una palabra código x, pero se recibe un vector y con, a lo más t errores. En este escenario, tenemos que $d(x, y) \leq t$. Ahora bien, si z es cualquier otra palabra código, entonces, debido a que $d(x, z) \leq d(x, y) + d(y, z)$, podemos afirmar que $d(y, z) \geq d(x, z) - d(x, y) \geq d - t$. Luego, dado que $t \leq \frac{d-1}{2}$ entonces $d \geq 2t + 1$ y por consiguiente tenemos que $d(y, z) \geq d - t \geq t + 1 > t$. Con lo cual concluimos que no hay otra palabra código cuya distancia al vector y sea menor que t. Por tanto, x es la palabra código más cercana a y en este caso. □

La siguiente definición nos permite interpretar la distancia mínima de un código lineal de una manera diferente.

Definición 1.2.3. *El peso de un vector $x \in \mathbb{F}_q^n$ es la cantidad de coordenadas no nulas o número de entradas no nulas del vector, es decir:*

$$\omega(x) = \#\{i : x_i \neq 0\},$$

por lo cual, el peso mínimo de un código está dado por

$$\omega_{\text{mín}}(C) = \text{mín}\{\omega(x) : x \in C, x \neq 0\}.$$

En un código lineal la distancia mínima d_{min} es el peso mínimo de las palabras del código distintas de cero.

Proposición 1.2.4. *Sea C un $[n, k, d]$ código lineal con distancia mínima d y peso mínimo ω. Entonces, $d_{min} = \omega_{min} = d$.*

Demostración. Sean C un $[n, k, d]$ código lineal y $x, y \in C$, note que $x - y \in C$ dado que C es subespacio vectorial. Luego,

$$d(x, y) = \#\{i : x_i \neq y_i\} = \#\{i : x_i - y_i \neq 0\}$$

$$\omega(x - y) = \#\{i : (x - y)_i \neq 0\} = \#\{i : x_i - y_i \neq 0\},$$

por lo tanto, $d_{min} = \omega_{min}$. □

Esta proposición tiene gran utilidad dado que nos proporciona una manera más rápida y sencilla de calcular la distancia mínima de un código lineal. Dado que para calcular la distancia mínima de un $[n, k]$ código lineal, por medio de la definición es necesario realizar $\begin{pmatrix} q^k \\ 2 \end{pmatrix}$ comparaciones para hallar la distancia entre cada par de palabras y después determinar el mínimo de las distancias,

mientras que si utilizamos la Proposición 1.2.4 solo es necesario calcular los pesos de $q^k - 1$ palabras no nulas del código y tomar el mínimo.

A continuación se presentan las funciones de SageMath que nos permiten obtener un código lineal aleatorio, la dimensión del código y su distancia mínima.

```
sage: C = codes.random_linear_code(base_field, length, dimension)
sage: C.dimension()
sage: C.minimum_distance()
```

Ejemplo 1. *Consideremos los siguientes códigos lienales*

```
sage: F = GF(2)
sage: C1=codes.random_linear_code(GF(2), 2, 1)
sage: list(C1)
[(0, 0), (0, 1)]
sage: C1.minimum_distance()
1
sage: C1.dimension()
1
sage: C2=codes.random_linear_code(GF(2), 3, 1)
sage: list(C2)
[(0, 0, 0), (1, 0, 1)]
sage: C2.minimum_distance()
2
sage: C2.dimension()
1
sage: C3=codes.random_linear_code(GF(2), 4, 1)
sage: list(C3)
[(0, 0, 0, 0), (0, 1, 1, 1)]
sage: C3.minimum_distance()
3
sage: C3.dimension()
1
```

El código $C1$ no puede detectar ni corregir errores. En contraste, el código $C2$ puede detectar errores, pero carece de la capacidad de corrección. Supongamos

que se envía la palabra $(1,0,1)$ *y se recibe* $(1,0,0)$*. En este caso, se puede determinar la existencia de un error, ya que la palabra recibida no pertenece al código. Sin embargo, identificar el error específico es problemático. Podríamos inferir incorrectamente que el error se encuentra en la primera coordenada, ajustando el mensaje a* $(0,0,0)$*, aunque esta corrección sería equivocada.*

En cuanto al código $C3$*, es capaz de detectar hasta 2 errores, pero solo puede corregir uno. Por ejemplo, si se envía la palabra* $x = (0,1,1,1)$ *y se recibe* $y = (0,1,0,1)$*, facilmente puede notarse que se produjo un error en la transmisión dado que la distancia de* x *a* y *es 1 pero la distancia mínima del código es de 3, este error puede corregirse fácilmente si consideramos la distancia que hay entre el vector recibido* y *a las palabras que forman parte del código para obtener la palabra más cercana a la recibida, en este caso tenemos* $d((0,0,0,0),y) = 2$ *y* $d((0,1,1,1),y) = 1$*, por lo que la palabra más cercana a la recibida es* $(0,1,1,1)$*. Por otro lado, si se presentan dos errores en la transmisión recibiendo el vector* $z = (0,0,0,1)$*, haciendo el mismo procedimiento anterior podríamos asumir erróneamente que el error cometido está en la última coordenada dado que la palabra más cernaca es* $(0,0,0,0)$*.*

Tal como se ha señalado previamente, los parámetros fundamentales de un código, a saber, longitud, dimensión y distancia mínima, desempeñan un papel de extrema importancia. Es una prioridad procurar que un código cuente con palabras de longitud reducida, así como una cantidad significativa de palabras y una distancia mínima considerablemente ”grande”, ya que la búsqueda de estos atributos en un código se traduce en una solución que optimiza el uso del espacio, expande las posibilidades de expresión y garantiza una transmisión fiable de información.

El siguiente ejemplo presenta un algoritmo desarrollado en SageMath que asegura la obtención de un código con una distancia mínima predefinida.

Ejemplo 2. *Para construir un código lineal* C *sobre el cuerpo finito de* q *elementos, longitud* n*, dimensión* k *y distancia mínima* d*, presentamos el siguiente programa.*

```
sage: def dist(q, n, k, d) :
    si=True
    while si :
```

```
        C=codes.random_linear_code(GF(q), n, k)
        if C.minimum_distance()==d :
            si=False
    return(C)
```

Usamos el código de SageMath anterior para obtener un código lineal C sobre $\mathbb{F}_2$, con parámetros longitud $n = 7$, dimensión $k = 4$ y distancia $d = 3$.

```
sage: C=dist(2, 7, 4, 3)
sage: C
[7, 4] linear code over GF(2)
```

Podemos comprobar que efectivamente el $[7, 4]$ código lineal obtenido tiene distancia mínima 3:

```
sage: C.minimum_distance()
3
```

Como la dimensión es $k = 4$ y el cuerpo empleado es $\mathbb{F}_2$ entonces tenemos que la cantidad de elementos del código es $2^4 = 16$, lo cual también podemos obtener mediante la función $C.cardinality()$ de SageMath. Por otro lado, la función $list(C)$ nos ayuda a obtener todas las palabras que hacen parte del código.

```
sage: C.cardinality()
16
sage: list(C)
[(0, 0, 0, 0, 0, 0, 0), (0, 0, 1, 1, 1, 0, 1), (1, 1, 1, 1, 1, 1, 1), (1, 1, 0, 0, 0, 1, 0),
(0, 1, 0, 1, 1, 0, 0), (0, 1, 1, 0, 0, 0, 1), (1, 0, 1, 0, 0, 1, 1), (1, 0, 0, 1, 1, 1, 0),
(0, 1, 0, 0, 1, 1, 1), (0, 1, 1, 1, 0, 1, 0), (1, 0, 1, 1, 0, 0, 0), (1, 0, 0, 0, 1, 0, 1),
(0, 0, 0, 1, 0, 1, 1), (0, 0, 1, 0, 1, 1, 0), (1, 1, 1, 0, 1, 0, 0), (1, 1, 0, 1, 0, 0, 1)]
```

Proposición 1.2.5. ***(Cota Singleton)*** *Sea C un $[n, k, d]$ código lineal sobre $\mathbb{F}_q$, entonces*

$$d \leq n - k + 1.$$

Demostración. Consideremos la proyección $\pi : C \to \mathbb{F}_q^{n-d+1}$ obtenida al eliminar $d-1$ coordenadas fijas, y como cada palabra de C tiene al menos d coordenadas no nulas, entonces π es inyectiva. Por lo tanto, $dim(\pi(C)) = k$ y $k \leq n-d+1$. □

Los códigos para los cuales se cumplen la igualdad en la cota de Singleton se denominan códigos de máxima distancia separable (MDS).

En el siguiente ejemplo haremos uso del algoritmo anterior para obtener un código lineal MDS.

Ejemplo 3. *Vamos a construir un código lineal sobre* $\mathbb{F}_2$ *con parámetros longitud* $n = 7$*, dimensión* $k = 3$ *y distancia mínima* $d = n - k + 1 = 7 - 3 + 1 = 5$.

```
sage: C=dist(2, 7, 3, 5)
sage: C
[7, 3] linear code over GF(2)
sage: C.minimum_distance()
5
```

Básicamene existen dos formas de describir un código lineal de dimensión k.

1. **Mediante una matriz generadora.**

   Definición 1.2.6. *Dada una base* $\mathcal{B} = \{v_1, \ldots, v_k\}$ *de un código lineal* C *se define una matriz generadora del código* C *como la matriz cuyas filas son los vectores* v_i *de la base.*

   Una matriz $G \in M_{k\times n}(\mathbb{F}_q)$ es una matriz generadora de un código lineal C si sus filas forman una base del espacio vectorial C, de modo que

   $$C = \{xG : x \in \mathbb{F}_q^k\}.$$

   G no está univocamente determinada por C, sino que depende la elección de la base. Recíprocamente, dada una matriz de orden $n \times k$ cuyas filas son linealmente independientes, existe un $[\mathrm{n}, \mathrm{k}]$ código para el cual esta matriz es la matriz generadora.

   Nótese que si dos matrices son equivalentes por filas entonces definen el mismo código, en este caso diremos que los códigos lineales son equivalentes.

   Para un $[\mathrm{n}, \mathrm{k}]$ código C sobre $\mathbb{F}_q$ podemos definir una forma de codificar mensajes usando una matriz generadora G. Los mensajes son vectores

arbitrarios de $\mathbb{F}_q^k$ y podemos codificarlos con la inyección de $\mathbb{F}_q^k$ en $\mathbb{F}_q^n$ que lleva un mensaje $u \in \mathbb{F}_q^k$ en una palabra código $c = uG \in \mathbb{F}_q^n$.

Ejemplo 4. *Sea C el código binario, es decir sobre F_2, de longitud 4 y dimensión 2 obtenido en SageMath y a continuación obtenemos su matriz generadora con el comando* ***.generator_matrix()***

sage: $codes.random\_linear\_code(GF(2), 4, 2)$
sage: $list(C)$
$[(0,0,0,0), (0,1,1,0), (1,1,1,1), (1,0,0,1)]$
sage: $C.generator\_matrix()$

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

sage: $C.minimum\_distance()$
2

El código binario C tiene una longitud de 4, una dimensión de 2 y una distancia mínima de 2 ya que su peso mínimo es 2. Esto implica que el código es capaz de detectar un error cometido durante la transmisión de información, pero no podrá corregirlo.

La ventaja de conocer una matrix generadora G de un $[n, k, d]$ código lineal C es que podemos obtener cada elemento del código de manera sencilla, calculando el producto de cada elemento de $\mathbb{F}_q^k$ por G. Por otro lado, dado que la matriz generadora de un código lineal no es única, algunas veces es conveniente trabajar con la matriz generadora estándar, la cual tiene la forma $(I_k|A)$, donde I_k es la matriz identidad $k \times k$ y $A \in M_{k \times n-k}$.

En el siguiente ejemplo mostramos las funciones en SageMath para obtener el código lineal C por medio de una matriz generadora y la función que nos proporciona la matriz generadora de un código lineal C.

Ejemplo 5. *Calcular*

a) *El código lineal sobre* $\mathbb{F}_2$ *generado por la matriz*

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

sage: $Matrix(GF(2), [[0,1,1,0,1,0,0],[0,0,1,1,0,1,0],$
$[0,0,0,1,1,0,1],[1,0,0,0,1,1,0]])$
sage: $C{=}LinearCode(G)$
sage: C
$[7,4]$ *linear code over* $GF(2)$
sage: $C.minimum\_distance()$
3
sage: $list(C)$
$[(0,0,0,0,0,0,0),(0,1,1,0,1,0,0),(0,0,1,1,0,1,0),(0,1,0,1,1,1,0),$
$(0,0,0,1,1,0,1),(0,1,1,1,0,0,1),(0,0,1,0,1,1,1),(0,1,0,0,0,1,1),$
$(1,0,0,0,1,1,0),(1,1,1,0,0,1,0),(1,0,1,1,1,0,0),(1,1,0,1,0,0,0),$
$(1,0,0,1,0,1,1),(1,1,1,1,1,1,1),(1,0,1,0,0,0,1),(1,1,0,0,1,0,1)]$

El código generado por la matriz G *es el* $[7,4,3]$ *código lineal sobre* $\mathbb{F}_2$. *Más adelante veremos que este código es equivalente al código de Hamming.*

b) *Su matriz generadora en forma estándar para el* $[7,4,3]$ *código lineal sobre* $GF(2)$ *anterior se obtiene con la siguiente función en SageMath*

sage: $C.systematic\_generator\_matrix()$

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

2. **Mediante una matriz de control de paridad.** Antes de definir la matriz de control de paridad de un código veamos la siguiente definición.

Definición 1.2.7. *Dado un* $[n,k]$ *código lineal* C *definimos el* ***código dual***

$C^\perp$ de C como

$$C^\perp = \{b \in \mathbb{F}_q^n : c \cdot b = 0, \forall c \in C\},$$

donde, $c \cdot b = \sum_{i=1}^n c_i b_i$ es el producto interno usual en $\mathbb{F}_q^n$.

Si se conoce la matriz generadora de C se puede caracterizar los elementos de $C^\perp$ como

$$C^\perp = \{x \in \mathbb{F}_q^n : x \cdot G^T = 0\} = \{x \in \mathbb{F}_q^n : G \cdot x^T = 0\}.$$

El código dual $C^\perp$ de un $[n, k]$ código lineal C es también un código lineal de parámetros $[n, n-k]$, por lo tanto admite una matriz generadora $H \in M_{(n-k)\times n}(\mathbb{F}_q)$.

Definición 1.2.8. *Una matriz H de orden $(n-k) \times n$ que es una matriz generadora de $C^\perp$ se llamará **matriz de control** o matriz de chequeo de paridad del código C.*

Claramente, una matriz de control de C es una matrix H de rango $n-k$ que satisface que para todo vector $x \in \mathbb{F}_q^n$, $x \in C$ si y solo si $Hx^T = 0$. Por lo tanto, esta matriz permite decidir (o controlar) si un vector está en el código o no.

Si la matriz generadora de un código C está dada en su forma estándar, es decir $G = (I_k|A)$ entonces una matriz de control para C es $H = (-A^T|I_{n-k})$, donde I_{n-k} es la matriz identidad de tamaño $n-k$. En efecto,

$$GH^T = (I_k|A)\begin{pmatrix} -A \\ I_{n-k} \end{pmatrix} = A - A = 0_{k\times(n-k)}.$$

Una matriz H de esta forma se dice que está en forma estándar como matriz de paridad aunque no está en forma estándar como matriz generadora de $C^\perp$.

Ejemplo 6. *Usando la siguiente función en SageMath obtenemos la matriz de control H del $[7, 4, 3]$ código lineal sobre $\mathbb{F}_2^7$ del Ejemplo 5.*

```
sage: H=C.parity_check_matrix()
sage: H
[1 0 0 1 0 1 1]
[0 1 0 1 1 1 0]
[0 0 1 0 1 1 1]
sage: C1=LinearCode(H)
sage: C1
[7, 3] linear code over GF(2)
```

Donde vemos que esta matriz no está de forma estándar como matriz de paridad, pero sí está en forma estándar como matriz generadora para $C^{\perp}$. La función que nos permite obtener el código generado a partir de una matriz de control H es $C = codes.from\_parity\_check\_matrix(H)$.

Teorema 1.2.9. *Sean C un $[n, k, d]$ código y H la matriz de control para C. Entonces*

$$d = \min\{r : \text{hay } r \text{ columnas linealmente dependientes en } H\}.$$

Es decir, H tiene d columnas linealmente dependientes pero cualquier conjunto de $d-1$ columnas son linealmente independientes.

Demostración. Consideremos $H_1, H_2, \ldots, H_n$ como las columnas de la matriz de control de paridad H. Entonces, para todo $c \in \mathbb{F}_2^n$ se tiene que $Hc^T = 0$, es decir, $c \in C$ sí, y solo si, $c_1H_1 + c_2H_2 + \cdots + c_nH_n = 0$. Ahora, si C es un código con peso mínimo d, entonces hay d columnas linealmente dependientes en H ya que si hay otra cantidad menor de columnas dependientes existiría una palabra no nula del código con peso menor a d. Recíprocamente, si hay r columnas linealmente dependientes entonces hay palabras de peso r y la distancia mínima es el menor de esos pesos. □

Ejemplo 7. *Verificamos que la matriz de paridad dada de forma estándar y la obtenida anteriormente generan el mismo código, el cual corresponde al código dual de C, $C^{\perp}$.*

```
sage: J=Matrix(GF(2), [[0, 1, 1, 1, 1, 0, 0], [1, 0, 1, 1, 0, 1, 0], [1, 1, 0, 1, 0, 0, 1]])
sage: J
```

```
[0 1 1 1 1 0 0]
[1 0 1 1 0 1 0]
[1 1 0 1 0 0 1]
```
sage: $C2 = LinearCode(J)$

sage: $C2$

$[7, 3]$ *linear code over* $GF(2)$

sage: $C1 == C2$

True

sage: $Ct = C.dual\_code()$

sage: $list(Ct)$

$[(0,0,0,0,0,0,0), (1,0,1,0,1,0,1), (0,1,1,0,0,1,1), (1,1,0,0,1,1,0),$
$(0,0,0,1,1,1,1), (1,0,1,1,0,1,0), (0,1,1,1,1,0,0), (1,1,0,1,0,0,1)]$

sage: $C1 == Ct$

$True$

Es de notar que también es una matriz de control ya que podemos observar que ambas están constituidas por las mismas filas pero ordenadas de distinta forma. La matriz J *corresponde a la matriz de control de paridad del código* C *dada en forma estándar, y en la línea 5 donde* $C1 == C2$ *verificamos que la matriz* J *y la matriz* H *generan el mismo* $[7, 3]$ *código lineal, además comprobamos que el código generado por dichas matrices es en efecto el código dual de* $C^{\perp}$*. Por otro lado, notemos que como* H *es una matriz de tamaño* 3×7 *entonces la longitud del código es 7 y la dimensión es 3. Además, como hay tres columnas linealmente dependientes y cualquier par de columnas son linealmente independientes, entonces la distancia mínima del código* C *es* $d = 3$*.*

A continuación se presenta una clase de códigos correctores de errores mejor conocidos como los códigos de Hamming. Esta es una familia de códigos de distancia mínima tres, es decir, estos códigos son capaces de detectar hasta dos errores y corregir un error.

1.3. CÓDIGOS DE HAMMING

El método Hamming es una técnica de red diseñada por Richard W. Hamming, para la detección de errores durante la transmisión de datos entre múltiples canales de red y para la corrección de los mismos. Los códigos de Hamming son fundamentales en la teoría de la codificación y cuentan con diversas aplicaciones prácticas. Tienen un papel crucial en nuestra vida cotidiana y son empleados en

una amplia variedad de dispositivos, como modems, memorias y sistemas de comunicaciones vía satélite.

Los códigos tradicionales de Hamming son códigos $[7, 4, 3]$ binarios, los cuales utilizan 7 bits para transmitir un bloque de datos de 4 bits junto con 3 bits de paridad para permitir la detección de dos errores y la corrección de uno de ellos.

Ejemplo 8. *En el* $[7, 4, 3]$ *código binario de Hamming, cada palabra del código tiene la forma:*

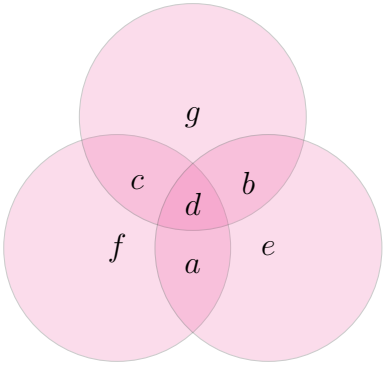
$$\underbrace{a \quad b \quad c \quad d}_{\text{Datos}} \quad \underbrace{e \quad f \quad g}_{\text{Paridad}},$$

en donde debe cumplirse que

$$\begin{aligned} a + b + d + e &= 0 \\ a + c + d + f &= 0 \\ b + c + d + g &= 0 \end{aligned}$$

En términos de diagrama de Venn se puede ver como:

Figura 1.2: Diagrama de Venn: $[7, 4, 3]$ código binario de Hamming.



Fuente: Elaboración propia.

En el diagrama, los símbolos de mensaje se ubican en el centro y se asegura que cada círculo contenga una cantidad par de “unos”(paridad uniforme).

El objetivo de los códigos de Hamming es crear un conjunto de bits de paridad que se superpongan para que se pueda detectar y corregir errores. Para permitir la corrección de errores, los códigos de Hamming utilizan varios bits de

paridad. Si el bloque de datos llega a su destino con una paridad diferente a la especificada, se deduce que al menos uno de los bits ha sido alterado. Estos códigos emplean múltiples bits de paridad para permitir la corrección de un error.

Definición 1.3.1. *Un código binario de Hamming* $H_2(r)$ *de longitud* $n = 2^r - 1$*, se define por su matriz de control cuyas filas consisten en todos los vectores binarios no nulos de longitud* r*. Esto nos da como resultado un código lineal con parámetros* $n = 2^r - 1$*,* $k = 2^r - r - 1$ *y* $d = 3$*.*

En SageMath la función para generar un código de Hamming es

sage: $C{=}codes.HammingCode(base\_field, order)$

Donde el parámetro $base\_field$ hace referencia al cuerpo finito sobre el cual se quiere construir el código, y el parámetro $order$ corresponde a la redundancia que queremos tenga el código.

Ejemplo 9. *Una matriz de control para el conocido* $[7, 4, 3]$ *código* $H_2(3)$ *es:*

sage: $C{=}codes.HammingCode(GF(2), 3)$
sage: C
$[7, 4]$ $Hamming\ Code\ over\ GF(2)$
sage: $C.parity\_check\_matrix()$

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

Donde podemos reconocer que las columnas de la matriz de control son las expresiones binarias de los números del 1 *al* 7*.*

Nota 1. *Todo código binario de parámetros* $[7, 4, 3]$ *es equivalente al* $[7, 4, 3]$ *código binario de Hamming, esto quiere decir que es el mismo código solo que aplicando una permutación de las coordenadas de sus palabras.*

Estas ideas de códigos binarios de Hamming pueden ser extendidas a códigos no binarios que tengan distancia mínima 3, es decir códigos correctores de un error.

1.3.1. Códigos de Hamming no binarios Ahora analicemos cómo extender la construcción de los códigos de Hamming a cualquier cuerpo $\mathbb{F}_q$, a estos códigos se les denomina códigos $q-$arios de Hamming, denotados $H_q(r)$ de longitud $n = (q^r - 1)/(q - 1)$. De manera similar a los códigos binarios de Hamming, el objetivo es garantizar que cualquier par de columnas en la matriz de control H sea linealmente independiente, lo que significa que ninguna columna de H sea un múltiplo de otra. Una forma fácil de construir una matriz de control para el $H_q(r)$ código de Hamming, es tomar todos los vectores no nulos de $\mathbb{F}_q^r$ tales que su primera coordenada no nula es 1. Esto dado que cada vector en $\mathbb{F}_q^r$ tiene $q^r - 1$ vectores no nulos, y el primer elemento tiene $q - 1$ opciones distintas de cero. Por otra parte, como no hay dos columnas que sean múltiplos entre sí, entonces este código tiene un peso mínimo de al menos 3.

Luego, el código asociado a esta matriz H de parámetros $[(q^r - 1)/(q - 1), (q^r - 1)/(q - 1) - r, 3]$, llamado código q-ario de Hamming tiene redundancia r y se denota $H_q(r)$.

Ejemplo 10. *Haciendo uso de la función* ***codes.HammingCode(base_field,order)*** *de SageMath, calculamos el* $H_3(2)$ *código no binario de Hamming*

sage: $C=codes.HammingCode(GF(3),2)$
sage: C
$[4, 2]$ $Hamming\ Code\ over\ GF(3)$
sage: $list(C)$
$[(0,0,0,0),(1,0,1,1),(2,0,2,2),(0,1,1,2),(1,1,2,0),(2,1,0,1),(0,2,2,1),$
$(1,2,0,2),(2,2,1,0)]$
sage: $C.parity\_check\_matrix()$
```
[1 0 1 1 0 1 0 1 1 1 0 1 1]
[0 1 1 2 0 0 1 1 2 0 1 1 2]
[0 0 0 0 1 1 1 1 1 2 2 2 2]
```
sage: $C.minimum\_distance()$
3
sage: $C.dimension()$
2
sage: $len(C)$
9

sage: $len(C[0])$
4

Podemos observar que las columnas de la matriz de control son, de hecho, vectores no nulos en $\mathbb{F}3$*, donde la primera coordenada no nula es* 1*. Además, es claro que la distancia mínima de este código es* 3*, la cual comprobamos con* $C.minimum\_distance()$*. La función* $len(C[0])$ *calcula la longitud de una de las palabras código, y nos indica que el código tiene longitud* 4*, mientras que por otro lado la función* $len(C)$ *determina la cantidad de palabras código. De esta manera, obtenemos el código* $[4, 2, 3]$ *de Hamming sobre el cuerpo* $\mathbb{F}_3$*.*

A continuación, se presentan los códigos Reed-Solomon, que se destacan debido a su impresionante capacidad de corrección de errores y su eficiencia computacional. Estos códigos constituyen una de las familias más ampliamente utilizadas en la actualidad, y sus extensiones tienen aplicaciones prácticas significativas. Son conocidos por su habilidad para corregir secuencias extensas de errores y símbolos faltantes, además de que las bases matemáticas subyacentes a estos códigos resultan igualmente interesantes.

1.4. CÓDIGOS REED-SOLOMON (RS)

Los códigos Reed-Solomon cuyo nombre deriva de I. Reed y G. Solomon quienes describieron estos códigos por primera vez en 1960, y tienen una amplia gama de aplicaciones en comunicaciones inalámbricas (teléfonos móviles, radio digital, televisión digital, entre otros) y en almacenamiento (DVD, códigos de barras, discos compactos, códigos QR, entre otros). Forman parte de los códigos correctores de errores más utilizados e importantes por su utilidad.

Definición 1.4.1. *Sean* $n \leq q-1$*,* $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_n)$ *con* $\alpha_i \in \mathbb{F}_q^*$ *y* $k \leqslant n$*, donde* k *corresponde al tamaño del mensaje. Consideremos el subconjunto de* $\mathbb{F}_q[x]$

$$L_k = \{p(x) \in \mathbb{F}_q[x] : grad(p(x)) < k\},$$

La tranformación lineal T *dada por*

$$\begin{aligned} T : & L_k \to \mathbb{F}_q^n \\ & p(x) \mapsto (p(\alpha_1), p(\alpha_2), \ldots, p(\alpha_n)). \end{aligned}$$

Dado que $p(x)$ tiene a lo sumo $k-1$ raíces y como $k < n$, entonces $k-1 < n-1$, luego $ker(T(L_k)) = \{0\}$. Por lo tanto, T es inyectiva.

Se define al código **Reed-Solomon** como $RS(\alpha) = Im(T(L_k))$.

Además nótese que como $RS(\alpha) = Im(T(L_k)) \subset \mathbb{F}_q^n$, tenemos que $T : L_k \to RS(\alpha)$ es biyectiva, y como L_k tiene dimensión k entonces RS tiene dimensión k.

Teorema 1.4.2. *El código Reed-Solomon tiene como distancia* $d = n - k + 1$.

Demostración. Sea $(a_0, a_1, \ldots, a_{k-1}) \neq 0$, el polinomio $p(x)$ tiene a lo sumo $k-1$ ceros y como $k < n$ implica que $(p(\alpha_1), p(\alpha_2), \ldots, p(\alpha_n))$ tiene a lo sumo $k-1$ ceros. Entonces $d = \omega(c) \geq n - (k-1)$ y por la cota Singleton, $d \leq n - k + 1$, por lo tanto $d = n - k + 1$.

□

Es decir, $RS(\alpha)$ es un código MDS, ya que alcanza la cota superior Singleton. Por lo tanto, $RS(\alpha) = Im(T)$ es un código lineal MDS con parámetros k y $d = n - k + 1$. En consecuencia, $RS(\alpha)$ es un $[n, k, n-k+1]$ código $q - ario$

En SageMath, podemos aprovechar la implementación incorporada para construir códigos Reed-Solomon de manera sencilla

sage: $C{=}codes.ReedSolomonCode(base\_field, length, dimension, primitive\_root = None)$

Donde el parámetro $base\_field$ hace referencia al cuerpo finito en el se quiere construir el código, $length$ corresponde a la longitud que queremos tenga el código, y dimension es la dimensión que queremos posea el código. Con respecto al parámetro $primitive\_root$, es completamente opcional. Si decidimos utilizarlo, debemos especificar un elemento primitivo del cuerpo en el que deseamos construir el código. Sin embargo, lo más común es no hacer uso de este parámetro y permitir que Sage seleccione automáticamente un elemento primitivo.

Ejemplo 11. *Ejemplo para obtener el código Reed-Solomon de logitud 8, dimension 2 y distancia mínima 7, usamos la siguiente función*

sage: C=$codes.ReedSolomonCode(GF(9), 8, 2)$
sage: C
$[8, 2, 7]$ $Reed - Solomon\ Code\ over\ GF(9)$.

1.4.1. Códigos Reed-Solomon Generalizados

Definición 1.4.3. *Sean n un entero tal que $1 \leq n \leq q$, k número entero con $1 \leq k \leq n$, $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_n)$, donde cada $\alpha_i \in \mathbb{F}_q$ son n elementos distintos, $v = (v_1, v_2, \ldots, v_n)$ donde los v_i (no necesariamente distintos) son elementos no nulos de $\mathbb{F}_q$. El código Reed-Solomon generalizado asociado a α y v se define como:*

$$RSG_k(\alpha, v) = \{(v_1 f(\alpha_1), v_2 f(\alpha_2), \ldots, v_n f(\alpha_n)) : f \in L_k\},$$

donde L_k es el espacio vectorial definido para los códigos RS.

Haciendo un análisis similar al utilizado para determinar los parámetros de los códigos RS, observamos que $RSG_k(\alpha, v)$ tiene dimensión k. Ahora bien, dado que un polinomio no nulo $f \in L_k$ tiene, como máximo, $k - 1$ ceros, $RSG_k(\alpha, v)$ tiene una distancia mínima de al menos $n - (k - 1) = n - k + 1$ y por la cota Singleton, tenemos $d \leq n - k + 1$, lo que implica que $RSG_k(\alpha, v)$ tiene una distancia mínima de $d = n - k + 1$. Por lo tanto, los códigos GRS también son MDS, al igual que los códigos RS. Los códigos RS, en sentido estricto son códigos RSG con $n = q - 1$, $\alpha = (\alpha_1, \ldots, \alpha_n)$, donde α es una raíz n-ésima primitiva de la unidad, y $v_i = (1, 1, \ldots, 1)$ para $0 \leq i \leq n - 1$. En consecuencia, GRS es un $[n, k, n - k + 1]$ código q-ario.

SageMath cuenta con la siguiente función que permite la construcción de códigos Reed-Solomon generalizados

sage: C=$codes.GeneralizedReedSolomonCode(evaluation\_points, dimension,$
$column\_multioliers$=$None)$

donde el parámetro $evaluation\_points$ es la lista de elementos distintos del cuerpo que se empleará, correspondiendo al vector α de la definición 1.4.3, el parámetro dimensión corresponde a la dimensión que deseamos tenga el código, y finalmente, el parámetro $column\_multioliers$ es el vector de elementos no nulos

del cuerpo que se esté empleando, correspondiente al vector v de la definición anterior. Este último parámetro no es obligatorio, sin embargo, si optamos por no emplearlo, SageMath asumirá automáticamente que el vector v que vamos a emplear tiene todas sus entradas con valor de 1. En otras palabras, si no se introduce este parámetro, el código que construiremos coincidirá con el código Reed-Solomon estándar.

Ejemplo 12. *Vamos a construir un código RSG de dimensión* 3 *y longitud* 7 *sobre* $\mathbb{F}_{11}$.

```
sage: q= 11
sage: k= 3
sage: a= vector(GF(q),[1,8,2,3,4,0,5])
sage: v= vector(GF(q),[2,4,6,1,3,5,6])
sage: C1= codes.GeneralizedReedSolomonCode(a,k,v)
sage: C1
[7, 3, 5] Generalized Reed-Solomon Code over GF(11).
```

Como deseamos construir un código de longitud 7 *lo que hacemos es tomar* a *como un vector de elementos dinstintos de* $\mathbb{F}_{11}^7$ *y* v *vector de elementos no nulos de* $\mathbb{F}_{11}^7$. *Para obtener el* $[7,3,5]$ *código generalizado Reed-Solomon sobre* $\mathbb{F}_{11}$.

El siguiente ejemplo presenta un programa desarrollado en SageMath que permite construir un código Reed-Solomon Generalizado (RSG) sobre el cuerpo finito $\mathbb{F}_q$, con una longitud l y una dimensión k de manera general. En este programa, se diseñan los vectores a y v de forma aleatoria, asegurando al mismo tiempo que se cumplan las condiciones definidas en la construcción del código RSG.

Ejemplo 13. *Códigos Reed-Solomon Generalizados en SageMath*

```
sage: def RSG(q,l,k):
....:     F = GF(q)
....:     long=l
....:     a=[]
....:     v=[]
....:     while len(a)<long:
....:         a_1=F.random_element()
```

```
...:         if a_1 not in a :
...:             a.append(a_1)
...:         print( 'a',vector(a))
...:     while len(v)<long :
...:         v_1=F.random_element()
...:         if v_1 !=0 :
...:             v.append(v_1)
...:         print( 'v',vector(v))
...:     C = codes.GeneralizedReedSolomonCode(a, k, b)
...:     return(C)
```

El programa comienza inicializando dos listas vacías, a y v, que se utilizan para almacenar los elementos de los vectores a y v. Estas listas se llenan gradualmente a medida que se inician los bucles $while$, donde se generan elementos aleatorios a_1 y v_1. Estos elementos se agregan a las listas a y v si a_1 no está previamente en a y si v_1 no es igual a cero, respectivamente. Este proceso se detiene cuando la longitud de cada lista alcanza la longitud deseada del código especificada previamente. Finalmente, se utilizan las listas α y v para crear un código Reed-Solomon Generalizado con los parámetros especificados anteriormente.

Si ejecutamos este programa con los mismos parámetros del ejemplo anterior, obtendremos un código con los mismos parámetros pero diferente al código que se presentó previamente

```
sage: RSG(11, 7, 3)
a = (5, 0, 3, 10, 6, 2, 9)
v = (2, 4, 6, 1, 3, 5, 6)
[7, 3, 5] Generalized Reed-Solomon Code over GF(11)
sage: C1==C
False
```

Queda claro que la elección de los vectores a y v en la construcción de los códigos Reed-Solomon generalizados es crucial y afecta las propiedades del código. Cambiar las coordenadas de estos vectores o generarlos de manera aleatoria dará como resultado códigos diferentes.

Por último queremos mencionar que esta familia de códigos RS y RSG aunque son códigos MDS, es decir maximizan los parámetros fundamentales del código lineal, tienen la desventaja de que códigos de longitud grande deben construirse sobre un cuerpo de característica grande también, dado que $n < q$. En la práctica, se necesitan códigos con logitud grande sobre un cuerpo de característica pequeña.

2. CÓDIGOS CÍCLICOS

Muchas familias de códigos lineales pueden ser vistos como códigos cíclicos, los cuales son unos de los códigos más estudiados y utilizados en la teoría de la información hoy en día, debido a la estructura algebraica que presentan. Incluyen los códigos de Golay, los códigos binarios de Hamming, y los códigos equivalentes a los códigos de Reed-Muller.

Definición 2.0.1. *Un código* $[n, k]$ *código lineal es cíclico si y solo si,* $\forall(c_0, c_1, \cdots, c_{n-1}) \in C$ *se tiene que* $(c_{n-1}, c_0, \cdots, c_{n-2}) \in C$.

Es decir, un código lineal C es cíclico, si es cerrado por el desplazamiento cíclico.

Ejemplo 14.

1. *El código de repetición, dado por:*

$$C = \{(0, 0, \cdots, 0), (1, 1, \cdots, 1), \cdots, (q-1, q-1, \cdots, q-1)\}$$

es un $[n, 1, n]$ *código cíclico.*

2. *El código de control de paridad:*

$$C = \{(c_1, c_2, \ldots, c_n) \in \mathbb{F}_q^n : \sum_{i=1}^{n} c_i = 0\}$$

es un $[n, n-1, 2]$ *código* q*-ario cíclico, puesto que para cualquier permutación de orden de los sumandos, la suma seguirá siendo* 0.

Una manera particular de representar los códigos cíclicos es en su forma polinomial, donde se representan las palabras código como polinomios en lugar de vectores. Esta representación facilita el análisis y la manipulación de los códigos cíclicos, ya que se pueden utilizar técnicas algebraicas y operaciones polinomiales para trabajar con ellos. A continuación, detallaremos el proceso de esta representación y cómo se lleva a cabo la conversión de palabras código en polinomios.

2.1. POLINOMIO GENERADOR

A continuación se presenta una caracterización de los códigos cíclicos, donde representaremos cada palabra del código como un polinomio en $\mathbb{F}_q[x]$ de grado a lo sumo $n-1$. Consideremos la aplicación φ, entre los espacios vectoriales $\mathbb{F}_q^n$ y $\mathbb{F}_q[x]$ dada por

$$\begin{aligned} \varphi : \mathbb{F}_q^n &\longrightarrow \mathbb{F}_q[x] \\ c = (c_0, \ldots, c_{n-1}) &\longmapsto c(x) = c_0 + c_1 x + \cdots + c_{n-1}x^{n-1}. \end{aligned}$$

Note que si $x^n = 1$ y $c(x) = c_0 + c_1 x + \cdots + c_{n-1}x^{n-1}$, entonces $xc(x) = c_{n-1} + c_0 x + \cdots + c_{n-2}x^{n-1}$, lo cual corresponde a un desplazamiento cíclico a la derecha de la palabra código c. De forma general tenemos que $x^m c(x) \in C$ representa un cambio cíclico de m coordenadas a la derecha de la palabra clave c. Así, si se multiplica cualquier polinomio del código por x módulo $x^n - 1$, obtenemos nuevamente otro polinomio del código. Por lo tanto, los códigos cíclicos de longitud n se pueden estudiar en el anillo $\mathbb{F}_q[x]$ modulo $x^n - 1$.

Sea $\langle x^n - 1\rangle$ el ideal generado por $x^n - 1$ y $A_n = \mathbb{F}_q[x]/\langle x^n - 1\rangle$.

El siguiente resultado nos permite identificar los códigos cíclicos como ideales del anillo cociente A_n.

Teorema 2.1.1. *Un $[n, k]$ codigo lineal C es cíclico si, y solo si visto en el anillo cociente A_n, es un ideal.*

Demostración. Supongamos que C es cíclico, queremos mostrar que C es un ideal de A_n, puesto que C es subgrupo abeliano de A_n basta mostrar que $\forall a(x) \in A_n$ y $c(x) \in C$ se tiene que $a(x)c(x) \in C$. Sea $a(x) \in A_n$ con $a(x) = a_0 + a_1 x + \cdots + a_{n-1}x^{n-1}$, entonces $a(x)c(x) = a_0 c(x) + a_1 x c(x) + \cdots + a_{n-1}x^{n-1}c(x)$, como cada $a_i c(x) \in C$ para cada $i = 0, \cdots n-1$ puesto que C es cerrado bajo la suma y el producto por escalares por ser subespacio, queda ver que $x^i c(x) \in C$, lo cual es claro dado que $x^i c(x)$ representa el $i-$ésimo cambio cíclico de c. Luego, C es un ideal de A_n.

Ahora, sea $c \in C$, con C ideal de A_n. Queremos ver que C es cíclico, como C ideal de A_n se tiene que C es subespacio vectorial de $\mathbb{F}_q^n$, luego, basta mostrar que $\forall (c_0, c_1, \ldots, c_{n-1}) \in C$ se tiene que $(c_{n-1}, c_0, \cdots, c_{n-2}) \in C$. Sea $(c_0, c_1, \ldots, c_{n-1}) \in$

C, entonces $cx = c_0x + \cdots + c_{n-1}x^n = c_{n-1} + c_0x + \cdots + c_{n-2}x^{n-1} \in C$ dado que $x \in A_n$ y C es un ideal. Es decir, $(c_{n-1}, c_0, \ldots, c_{n-2}) \in C$. Por lo tanto, C es cíclico. □

Dado que A_n es un dominio de ideales principales, es decir, todo ideal de A_n es principal entonces todo código cíclico es generado por un polinomio mónico $g(x)$ de menor grado en C.

Denotaremos como $g(x)$ al polinomio generador del código C y al código generado por $g(x)$ como

$$C = \langle g(x) \rangle = \{g(x) \cdot q(x) : q(x) \in A_n\}.$$

Determinar qué aspecto debe tener un polinomio generador es un factor de gran importancia, en particular, conocer su grado. El grado de un polinomio generador en la representación polinomial de un código cíclico es esencial, ya que determina la longitud máxima del ciclo que puede detectar y corregir el código. Para abordar esta cuestión, presentamos el siguiente teorema.

Teorema 2.1.2. *Sea C un $[n, k]$ código cíclico distinto de cero de longitud n en A_n, entonces*

1. *Existe un único polinomio mónico de grado mínimo $g(x)$ en C*
2. *$g(x)$ es el polinomio mónico generador de C, es decir, $C = \langle g(x) \rangle$*
3. *$g(x)|(x^n - 1)$*

Demostración. Veamos por separado los distintos apartados del teorema.

1. Primero mostraremos la existencia del polinomio mónico $g(x)$. Consideremos $R = \{grad(g(x))|g(x) \in C\}$, nótese que $R \neq \emptyset$ y $R \subseteq \mathbb{N}$, entonces por el principio del buen orden de $\mathbb{N}$ tenemos que existe elemento mínimo $r > 0$ tal que $r = grad(g_1(x))$, con $g_1(x) = c_1 + c_2x + \cdots c_rx^r$, por lo cual, basta considerar $g(x) = c_r^{-1} \cdot g_1(x)$. Ahora, veamos la unicidad de $g(x)$. Sean $g_1(x)$ y $g_2(x)$ polinomios mónicos de C de grado mínimo r, nótese que $g_1(x) - g_2(x) \in C \setminus \{0\}$ pues $g_1(x) \neq g_2(x)$ y por ser C un ideal. De este modo tenemos $grad(g_1(x) - g_2(x)) < r$, lo que contradice la minimalidad de $g_1(x)$ y $g_2(x)$. Por tanto, existe un único polinomio mónico de grado mínimo $g(x)$ en C.

2. Veamos que $C \subseteq \langle g(x)\rangle$. Sea $c(x) \in C$, por el algoritmo de la división tenemos que existen $q(x), r(x) \in \mathbb{F}_q[x]$ tales que $c(x) = g(x)q(x) + r(x)$, con $grad(r(x)) < grad(g(x))$ ó $r(x) = 0$, luego, como $c(x) - g(x)q(x) = r(x) \in C$ y el grado de $r(x)$ no puede ser menor que el grado de $g(x)$, entonces $r(x) = 0$. Por lo cual, $c(x)$ es un múltiplo de $g(x)$. Es decir, $c(x) \in \langle g(x)\rangle$. Ahora veamos que $\langle g(x)\rangle \subseteq C(x)$. Recordemos que $\langle g(x)\rangle = \{g(x)q(x) : q(x) \in A_n\}$, como C es un ideal de A_n, si $g(x) \in C$, entonces $\langle g(x)\rangle \subset C$.

3. Supongamos que $g(x) \nmid x^n - 1$, entonces $x^n - 1 = g(x)q(x) + r(x)$ con $0 < grad(r(x)) < grad(g(x))$, y como $r(x) = -g(x)q(x)$ en A_n entonces $r(x) \in \langle g(x)\rangle$, lo cual contradice la minimalidad de $g(x)$. Por lo tanto, $g(x) \mid x^n - 1$.

□

A continuación se prensenta un teorema que nos permitirá la deducción de la existencia de una correspondencia biyectiva entre el conjunto de polinomios mónicos que dividen a $x^n - 1$ sobre $\mathbb{F}_q[x]$ y los códigos cíclicos en $\mathbb{F}_q^n$.

Teorema 2.1.3. *Un polinomio mónico $p(x) \in A_n$ es el polinomio generador de un código cíclico de longitud n sobre $\mathbb{F}_q$ si y sólo si $p(x) \mid x^n - 1$.*

Demostración. $\Rightarrow$] Esta implicación es consecuencia del Teorema 2.1.2.
$\Leftarrow$] Si $C = \langle p(x)\rangle$, queremos mostrar que $p(x)$ es de grado mínimo. Supongamos que $p(x) \mid x^n - 1$ y sea $g(x)$ el polinomio generador de $C = \langle p(x)\rangle$, tal que $p(x) \neq g(x)$. Como $p(x)$ y $g(x)$ son mónicos, y $g(x)$ es de grado mínimo, entonces $\mathrm{grad}(g(x)) \leq \mathrm{grad}(p(x))$. Por hipótesis, $x^n - 1 = p(x) \cdot q(x)$ para algún polinomio $q(x) \neq 0$.

Por otro lado tenemos que $g(x) \in C$ y aís, $g(x) \in \langle p(x)\rangle$ $i.e$, existe $a(x) \in A_n$ tal que $g(x) = a(x) \cdot p(x)$. De manera que:

$$g(x) \cdot f(x) = a(x) \cdot p(x) \cdot f(x) = a(x) \cdot (x^n - 1)$$

Por lo tanto, $\mathrm{grad}(g(x) \cdot f(x)) \geq \mathrm{grad}\,(x^n - 1) = \mathrm{grad}(p(x) \cdot f(x))$, esto quiere decir que $\mathrm{grad}(g(x)) \geq \mathrm{grad}(p(x))$. Por tanto, $\mathrm{grad}(g(x)) = \mathrm{grad}(p(x))$ y, como $p(x)$ y $g(x)$ son polinomios mónicos, tienen que ser iguales, $p(x) = g(x)$. Luego, $p(x)$ es el polinomio generador de C. □

Corolario 2.1.4. *Sobre $\mathbb{F}_q$ existen tantos códigos cíclicos de longitud n como divisores mónicos del polinomio $x^n - 1$.*

Este corolario nos permite apreciar la importancia que tiene el poder factorizar el polinomio $x^n - 1$ en polinomios mónicos irreducibles, esto con el fin de encontrar todos los códigos cíclicos de longitud n sobre $\mathbb{F}_q$. Pues, nótese que si podemos factorizar $x^n - 1 = p_1(x)p_2(x)\cdots p_m(x)$, donde cada $p_i(x)$ es un polinomio mónimo irreducible sobre $\mathbb{F}_q[x]$, entonces cada subconjunto $M \subseteq \{1, \ldots, m\}$ produce un código cíclico de longitud n dado por

$$C_M = \langle g_M(x) \rangle \text{ con, } g_M(x) = \prod_{i \in M} p_i(x).$$

En SageMath la función $codes.CyclicCode(generator\_pol = g, length = n)$ nos permite obtener el código cíclico a partir de un polinomio generador, especificando su polinomio generador g y la longitud que se requiere.

Una forma de generar todos los códigos cíclicos de longitud n sobre el cuerpo finito de q elementos es usando la siguiente función en SageMath.

```
sage: def AllCiclic(n, q) :
....:     F.<x> = GF(q)[ ]
....:     D=divisors(x^n - 1)
....:         l=len(D)
....:         for i in range (0, l) :
....:             C=codes.CyclicCode(generator_pol=D[i], length=n)
....:         print(C)
....:         print( 'Polinomio generador :',D [i])
```

A continuación se presenta un ejemplo clásico que nos permite visualizar cómo calcular todos los códigos cíclicos binarios de longitud 7.

Ejemplo 15. *Encontrar todos los códigos cíclicos sobre* $\mathbb{F}_2^7$.

Primero debemos descomponer $x^7 - 1$ en polinomios mónicos irreducibles en $\mathbb{F}_2[x]$. No es difícil verificar que dicha factorización es:

$$x^7 - 1 = (x + 1)(x^3 + x + 1)(x^3 + x^2 + 1) = p_1(x)p_2(x)p_3(x)$$

Por lo que entonces hay 2^3 códigos cícliclos distintos, de longitud 7. Ahora, como ya tenemos los divisores de $x^7 - 1$, podemos determinar los polinomios generadores de cada código cíclico.

$g_1(x) = 1$
$g_2(x) = p_1(x) = x + 1$
$g_3(x) = p_2(x) = x^3 + x + 1$
$g_4(x) = p_3(x) = x^3 + x^2 + 1$
$g_5(x) = p_1(x)p_2(x) = (x + 1)(x^3 + x + 1) = x^4 + x^2 + x + 1$
$g_6(x) = p_1(x)p_3(x) = (x + 1)(x^3 + x^2 + 1) = x^4 + x^3 + x^2 + 1$
$g_7(x) = p_2(x)p_3(x) = (x^3 + x + 1)(x^3 + x^2 + 1) = x^6 + x^5 + x^4 + x^3 + x^2 + x + 1$
$g_8(x) = p_1(x)p_2(x)p_3(x) = x^7 - 1$

Por lo que sus respectivos códigos cíclicos son:

Cuadro 2.1: Códigos cíclicos de longitud 7.

Polinomio generador	Parámetros	Código generado
$C_1 = \mathbb{F}_2[x]/(x^7 - 1) = \mathbb{F}_2^7$	$[7, 7]$	$\mathbb{F}_2^7$
$C_2 = \langle x + 1 \rangle$	$[7, 6]$	Código de control de paridad
$C_3 = \langle x^3 + x + 1 \rangle$	$[7, 4]$	Código de Hamming
$C_4 = \langle x^3 + x^2 + 1 \rangle$	$[7, 4]$	Código de Hamming
$C_5 = \langle x^4 + x^2 + x + 1 \rangle$	$[7, 3]$	Dual del código de Hamming C_4
$C_6 = \langle x^4 + x^3 + x^2 + 1 \rangle$	$[7, 3]$	Dual del código de Hamming C_3
$C_7 = \langle x^6 + x^5 + x^4 + x^3 + x^2 + x + 1 \rangle$	$[7, 1]$	Código de repetición
$C_8 = 0$	$[7, 0]$	Código nulo

Fuente: Elaboración propia.

Un ejemplo de uno de los códigos cíclicos escritos de manera completa es:

$$\begin{aligned} C_5 = \langle x^4 + x^3 + x^2 + 1 \rangle &= \{(x^4 + x^3 + x^2 + 1)q(x) : grad(q(x)) < 3\} \\ &= \{(x^4 + x^3 + x^2 + 1)(a + bx + cx^2) : a, b, c \in \mathbb{F}_2\}. \end{aligned}$$

Entonces tenemos los siguientes elementos que conforman a C_5:
$(x^4 + x^3 + x^2 + 1) \cdot 1 = \{x^4 + x^3 + x^2 + 1\}$
$(x^4 + x^3 + x^2 + 1)x = \{x + x^3 + x^4 + x^5\}$
$(x^4 + x^3 + x^2 + 1)(x + 1) = \{x^5 + x^2 + x + 1\}$
$(x^4 + x^3 + x^2 + 1)(x^2) = \{x^6 + x^5 + x^4 + x^2\}$
$(x^4 + x^3 + x^2 + 1)(x^2 + x) = \{x^6 + x^3 + x^2 + x\}$
$(x^4 + x^3 + x^2 + 1)(x^2 + 1) = \{x^6 + x^5 + x^3 + 1\}$
$(x^4 + x^3 + x^2 + 1)(x^2 + x + 1) = \{x^6 + x^4 + x + 1\}$

dichas representaciones polinomiales corresponden a los siguientes elementos del código:

$$C_5 = \{(1,0,1,1,1,0,0),(0,1,0,1,1,1,0),(1,1,1,0,0,1,0),(0,0,1,0,1,1,1),$$

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

Usando las funciones de SageMath descritas anteriormente, obtenemos el código cíclico generado por el polinomio $g = x^4 + x^3 + x^2 + 1$ de longitud 7 y cada palabra código que lo conforma.

sage: $F.\langle x\rangle = GF(2)[\,]$
sage: $n{=}7$
sage: $g{=}x^4{+}x^3{+}x^2{+}1$
sage: $C{=}codes.CyclicCode(generator\_pol{=}g, length{=}n)$
sage: C
$[7,3]\ Cyclic\ Code\ over\ GF(2)$
sage: $list(C)$
$[(0,0,0,0,0,0,0),(1,0,1,1,1,0,0),(0,1,0,1,1,1,0),(1,1,1,0,0,1,0),$
$(0,0,1,0,1,1,1),(1,0,0,1,0,1,1),(0,1,1,1,0,0,1),(1,1,0,0,1,0,1)]$

El siguiente resultado nos permite obtener la matriz generadora y la dimensión de un código cíclico a partir de su polinomio generador.

Teorema 2.1.5. *Sea C un $[n,k]$ código cíclico con polinomio generador, $g(x) = g_0 + g_1x + \cdots + g_rx^r$. Entonces, $g_0 \neq 0$ y $\{g(x), xg(x), \cdots, x^{n-r-1}g(x)\}$ es una base para C.*
En consecuencia, $k = dim(C) = n - grad(g(x))$ y G es una matriz generadora,

$$G = \begin{bmatrix} g_0 & g_1 & \cdots & g_r & 0 & \cdots & 0 \\ 0 & g_0 & \cdots & g_{r-1} & g_r & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & g_0 & g_1 & \cdots & g_r \end{bmatrix}.$$

Demostración. Supongamos $g_0 = 0$, entonces $x^{n-1}g(x) = g_1 + g_2x + \cdots + g_r^{r-1}x^{r-1}$, lo cual contradice la minimalidad de $g(x)$. Por otra parte, como $g_0 \neq 0$, entonces $g(x), xg(x), \cdots, x^{n-r-1}g(x)$ son linealmente independitente. Luego, sea $c(x) = c_0 + c_1x + \cdots + c_{n-1}x^{n-1} \in C$, entonces existe $q(x) = q_0 + q_1x + \cdots q_{n-r-1}x^{n-r-1} \in A_n$ tal que $c(x) = q(x)g(x) = q_0g(x) + q_1xg(x) + \cdots + q_{n-r-1}x^{n-r-1}g(x)$, obteniendo que

cada $c(x) \in C$ es combinación lineal de $\{g(x), xg(x), \cdots, x^{n-r-1}g(x)\}$. Por tanto, $k = dim(C) = n - grad(g(x))$ y G es una matriz generadora,

$$G = \begin{bmatrix} g_0 & g_1 & \cdots & g_r & 0 & \cdots & 0 \\ 0 & g_0 & \cdots & g_{r-1} & g_r & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & g_0 & g_1 & \cdots & g_r \end{bmatrix}.$$

□

Ejemplo 16. *Consideremos el código binario cíclico de longitud 7 con polinomio generador $g(x) = x^3 + x + 1$, entonces aplicando el teorema anterior, una matriz generadora para este código viene dada por*

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

Es de notar que la matriz generadora G es de tamaño 4×7 y rango 4, siendo 4 la dimensión del código C.

En el siguiente ejemplo, utilizaremos las herramientas de SageMath que hemos presentado previamente para encontrar el polinomio y la matriz generadora de un código cíclico, con el objetivo de verificar el Teorema 2.1.5.

Ejemplo 17. *Consideremos el siguiente código cíclico obtenido en SageMath.*

sage: $F.\langle a \rangle = GF(3)[\,]$
sage: $n=8$
sage: $Cc=codes.CyclicCode(length=n, field=F, D=[1,2])$
sage: Cc
$[8,4]\ Cyclic\ Code\ over\ GF(3)$
sage: $Cc.generator\_polynomial()$
$x^4 + 2x^3 + 2x + 2$
sage: $Cc.generator\_matrix()$

```
[2 2 0 2 1 0 0 0]
[0 2 2 0 2 1 0 0]
[0 0 2 2 0 2 1 0]
[0 0 0 2 2 0 2 1]
```

sage: $Cc.minimum\_distance()$
4

Obtenemos el $[8, 4, 4]$ *código cíclico sobre* $\mathbb{F}_3$*, polinomio generador* $g(x) = x^4 + 2x^3 + 2x + 2$ *y la matriz generadora verifica el Teorema 2.1.5 con* $g_0 = 2, g_1 = 2, g_2 = 0, g_3 = 2$ *y* $g_4 = 1$.

El siguiente resultado nos permite definir el polinomio de control de un código cíclico, con el cual se nos facilitará obtener una matriz de control para el código C.

Lema 2.1.6. $h(x) = \frac{x^n-1}{g(x)} = h_0 + \cdots + h_k x^k$ *es un polinomio de control de paridad para* C*. Es decir:*

$$c(x) \in C \Leftrightarrow h(x)c(x) = 0_{A_n}$$

Demostración. $\Rightarrow]$ Sea $c(x) \in C$. Luego, $c(x) = f(x)g(x)$, y así $h(x)c(x) = f(x)(h(x)g(x)) = f(x)(x^n - 1) = 0$ en A_n.

$\Leftarrow]$ Si $c(x) \in C$ es tal que $h(x)c(x) = 0$ en A_n, entonces $h(x)c(x) = f(x)(x^n - 2)$, con $f(x) \in \mathbb{F}_q[x]$. Por tanto, $h(x)c(x) = f(x)(h(x)g(x))$ y así, $c(x) = f(x)g(x)$, pues $\mathbb{F}_q[x]$ es dominio entero. De la última expresión, sigue que $c(x) \in C$.
Por lo tanto, $c(x) \in C \Leftrightarrow h(x)c(x) = 0 \in A_n$.

□

Definición 2.1.7. ***Polinomio de control de*** C*). Sea* $g(x)$ *el polinomio de grado* r *generador del código cíclico* C*. Como* $g(x)$ *divide a* $x^n - 1$ *entonces*

$$g(x)h(x) = x^n - 1,$$

con $h(x)$ *polinomio de grado* $n - r$*, a este polinomio se le llama polinomio de control de* C*.*

La función en SageMath que nos permite hallar el polinomio de control de un código cíclico C es

sage: $C.check\_polynomial()$

Ejemplo 18. *Calcularemos el polinomio de control del código cíclico de longitud* 5 *generado por* $g(x) = x + 1$

sage: $F.\langle x\rangle = GF(2)[\,]$
sage: $C{=}codes.CyclicCode(x+1, 5)$
sage: C
$[5, 4]$ *Cyclic Code over* $GF(2)$
sage: $C.check\_polynomial()$
$x^4 + x^3 + x^2 + x + 1$

y comprobamos que en efecto $h(x)g(x) = (x^4 + x^3 + x^2 + x + 1)(x + 1) = x^5 + 1$.

Las características de este polinomio se presentan en el siguiente resultado.

Teorema 2.1.8. *Sean* C *un código cíclico y* $h(x)$ *su polinomio de control, entonces*

1. $C = \{c(x) \in A_n : c(x)h(x) = 0\}$.
2. *Si* $h(x) = h_0 + h_1x + \cdots + h_{n-r}x^{n-r}$, *entonces la matriz de control de* C *está dada por*
$$\begin{pmatrix} h_{n-r} & \cdots & h_0 & 0 & \cdots & \cdots & 0 \\ 0 & h_{n-r} & \cdots & h_0 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \cdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & h_{n-r} & \cdots & h_0 & 0 \\ 0 & \cdots & \cdots & 0 & h_{n-r} & \cdots & h_0 \end{pmatrix}.$$
3. *El código dual* $C^\perp$ *es el código cíclico de dimensión* r *con polinomio generador*
$$h^\perp(x) = x^{n-r}h\left(x^{-1}\right) = \left(h_0x^{n-r} + h_1x^{n-r-1} + \cdots + h_{n-r}\right).$$
4. *El código generado por* $h(x)$ *corresponde al código dual con coordenadas invertidas, este código será denotado por* $\overleftarrow{C^\perp}$.

Demostración. 1. Sea $c(x) \in C$. Entonces $c(x) = f(x)g(x)$, para algún $f(x) \in A_n$. Por tanto, $c(x)h(x) = f(x)g(x)h(x) = f(x)(x^n - 1) = 0$. Recíprocamente, sea $c(x) \in A_n$ tal que $c(x)h(x) = 0$. Por el algoritmo de la división tenemos $c(x) = f(x)g(x) + r(x)$, con $grad(r(x)) < r$ o $r(x) = 0$. Luego,

$$c(x)h(x) = f(x)g(x)h(x) + r(x)h(x) = r(x)h(x)$$

como $grad(r(x)h(x)) < r + (n - r) = n$ se tiene que $r(x)h(x) = 0$, es decir, $r(x) = 0$, y por consiguiente $c(x) = f(x)g(x)$, en otras palabras, $c(x) \in C$.

2. Si $c(x) \in C$, entonces $c(x)h(x) \equiv 0$ con $grad(c(x)) = r < n$ y $grad(h(x)) = n-r$. Así, $grad(c(x)h(x)) < n+n-r = 2n-r$, con lo cual sobre $\mathbb{F}_q[x]$ se tiene $c(x)h(x) = (x^n-1)f(x)$ con $grad(f(x)) < n-r$. Luego $c(x)h(x) = (x^n-1)(f_0+f_1x+\cdots+f_{n-r-1}x^{n-r-1} = f_0x^n+f_1x^{n+1}+\cdots+f_{n-r-1}x^{2n-r-1}+(f_0+f_1x+\cdots+f_{n-r-1}x^{n-r-1})$, vemos que los coeficientes de $x^{n-r}, x^{n-r+1}, \ldots, x^{n-1}$ en el producto $c(x)h(x)$ son cero, es decir, tenemos el siguiente sistema de ecuaciones

$$\begin{aligned} c_0h_{n-r} + c_1h_{n-r-1} + \cdots + c_{n-r}h_0 &= 0 \\ c_1h_{n-r} + c_2h_{n-r-1} + \cdots + c_{n-r+1}h_0 &= 0 \\ c_{r-1}h_{n-r} + c_rh_{n-r-1} + \cdots + c_{n-1}h_0 &= 0. \end{aligned}$$

lo cual es equivalente a $(c_0c_1\ldots c_{n-1})\,H^T = 0$, es decir, H genera un código C' que es ortogonal a C, o sea, $C' \subseteq C^\perp$. Además, como $h_{n-r} \neq 0$, se tiene que $\dim C' = r$ y por lo tanto $C' = C^\perp$.

3. Como $h(x)g(x) = x^n - 1$ entonces $h\left(x^{-1}\right) g\left(x^{-1}\right) = x^{-n} - 1$, así,

$$x^{n-r}h\left(x^{-1}\right) x^r g\left(x^{-1}\right) = 1 - x^n,$$

es decir, el polinomio mónico $g(x)^\perp = h_0^{-1}x^{n-r}h\left(x^{-1}\right)$ divide a $x^n - 1$. Luego, el código cíclico generado por $g^\perp(x)$ tiene como matriz generadora a H y a su vez H es la matriz de control de C, por lo tanto, $\langle g^\perp(x)\rangle = C^\perp$.

4. Notemos que de la Definición 2.1.7 se puede concluir que $h(x)$ genera un código cíclico de longitud n y que además este código está conformado por las coordenadas invertidas del código dual C, pues el polinomio generador $h(x) = h_0 + h_1x + \cdots + h_{n-r}x^{n-r}$ es el polinomio invertido del polinomio generador de $C^\perp$ $h^\perp(x) = (h_0x^{n-r} + h_1x^{n-r-1} + \cdots + h_{n-r})$. Al código generado por el polinomio de control le denotaremos $\overleftarrow{C^\perp}$. □

Ejemplo 19. *Consideremos el código cíclico C de longitud 7 generado por el polinomio $g(x) = x^4 + x^2 + x + 1$, y usando la función $codes.CyclicCode(generator\_pol = g, length = n)$ de SageMath obtenemos el $[7,3]$ código cíclico sobre $\mathbb{F}_2^7$, y con ayuda de la función $C.check\_polynomial()$ obtenemos el polinomoio de control $h(x) = x^3 + x + 1$*

```
sage: F.<x> = GF(2)[]
sage: C=codes.CyclicCode(x^4 + x^2 + x + 1, 7)
```

sage: C
$[7, 3]$ *Cyclic Code over* $GF(2)$
sage: $C.minimum\_distance()$
4
sage: $C.check\_polynomial()$
$x^3 + x + 1$
sage: $H{=}C.parity\_check\_matrix()$
sage: H

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

Una vez conocido el polinomio de control de C podemos hallar el polinomio generador de $C^{\perp}$

$$g^{\perp}(x) = x^{n-r}h(x^{-1}) = x^{7-4}(x^{-3} + x^{-1} + 1) = x^3(x^{-3} + x^{-1} + 1) = 1 + x^2 + x^3.$$

Además, nótese que como $h(x) = 1+x+x^3$, la matriz H de control de C obtenida en SageMath satisface el teorema anterior, con $h_0 = 1, h_1 = 1, h_2 = 0$ y $h_3 = 1$

$$\begin{pmatrix} h_{7-4} & h_{7-5} & h_{7-6} & h_{7-7} & 0 & 0 & 0 \\ 0 & h_{7-4} & h_{7-5} & h_{7-6} & h_{7-7} & 0 & 0 \\ 0 & 0 & h_{7-4} & h_{7-5} & h_{7-6} & h_{7-7} & 0 \\ 0 & 0 & 0 & h_{7-4} & h_{7-5} & h_{7-6} & h_{7-7} \end{pmatrix} = \begin{pmatrix} h_3 & h_2 & h_1 & h_0 & 0 & 0 & 0 \\ 0 & h_3 & h_2 & h_1 & h_0 & 0 & 0 \\ 0 & 0 & h_3 & h_2 & h_1 & h_0 & 0 \\ 0 & 0 & 0 & h_3 & h_2 & h_1 & h_0 \end{pmatrix}$$
$$= \begin{pmatrix} 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix},$$

siendo $r = 4$ el grado del polinomio generador de C.

Ejemplo 20. *Al considerar el* $[8, 4, 4]$ *código cíclico sobre* $\mathbb{F}_3$ *del Ejemplo 17, obtenemos que su polinomio de control está dado por:*

sage: $Cc.check\_polynomial()$
$x^4 + x^3 + x^2 + 2x + 1$

El siguiente ejemplo ilustra la obtención de la matriz de control de un código lineal C a partir de su polinomio de control.

Ejemplo 21. *Consideremos C el código binario cíclico de longitud 7 con polinomio generador $g(x) = x^3 + x + 1$, entonces su polinomio de control es $h(x) = x^4 + x^2 + x + 1$. Eplicando la Proposición 2.1.8, una matriz de control de C viene dada por*

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

Verificamos este resultado en SageMath a continuación

sage: $F.\langle x \rangle = GF(2)[\,]$
sage: $n=7$
sage: $g=x^3+x+1$
sage: $C=codes.CyclicCode(generator\_pol=g, length=n)$
sage: C
$[7, 4]$ *Cyclic Code over* $GF(2)$
sage: $C.check\_polynomial()$
$x^4 + x^2 + x + 1$
sage: $C.parity\_check\_matrix()$

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

Nótese que la matriz de control H es de tamaño 3×7 y con rango 3, esta característica se deriva del hecho de que la codificación C posee dimensión 4. Además, esta matriz genera al código dual $C^\perp$ de C, que es también cíclico con polinomio generador $h(x) = 1 + x^2 + x^3 + x^4$, que es precisamente el polinomio $g_5(x)$ del Ejemplo 15, en el que se calculaban todos los polinomios generadores de códigos cíclicos binarios de longitud 7. Es importante resaltar con este ejemplo que el polinomio de control $h(x)$ de un código cíclico C no es necesariamente el polinomio generador de $C^\perp$, aunque tenga el grado adecuado para generar $C^\perp$, sea mónico y divisor de $x^n - 1$. Verificamos esto en SageMath a continuación

sage: $Ct{=}C.dual\_code()$
sage: Ct
$[7, 3]\ Cyclic\ Code\ over\ GF(2)$
sage: $Ct1{=}codes.CyclicCode(x^4 + x^2 + x + 1, 7)$
sage: $Ct1$
$[7, 3]\ Cyclic\ Code\ over\ GF(2)$
sage: $Ct{==}Ct1$
$False$
sage: $list(Ct)$
$[(0, 0, 0, 0, 0, 0, 0), (1, 0, 1, 1, 1, 0, 0), (0, 1, 0, 1, 1, 1, 0), (1, 1, 1, 0, 0, 1, 0),$
$(0, 0, 1, 0, 1, 1, 1), (1, 0, 0, 1, 0, 1, 1), (0, 1, 1, 1, 0, 0, 1), (1, 1, 0, 0, 1, 0, 1)]$
sage: $list(Ct1)$
$[(0, 0, 0, 0, 0, 0, 0), (1, 1, 1, 0, 1, 0, 0), (0, 1, 1, 1, 0, 1, 0), (1, 0, 0, 1, 1, 1, 0),$
$(0, 0, 1, 1, 1, 0, 1), (1, 1, 0, 1, 0, 0, 1), (0, 1, 0, 0, 1, 1, 1), (1, 0, 1, 0, 0, 1, 1)]$

Calculando el código dual del código C con la función $C.dual\_code()$ y el código cíclico $Ct1$ generado por el polinomio de control $g(x) = x^4 + x^2 + +1$ sobre $\mathbb{F}_2$, obtenemos que $Ct \neq Ct1$.

A continuación se presenta un resultado de códigos cíclicos sobre cuerpos finitos donde la estructura algebraica del código está vinculada a las raíces de la unidad.

2.2. RAÍCES n-ÉSIMAS DE LA UNIDAD Y CLASES CICLOTÓMICAS

Definición 2.2.1. ***Clases q-ciclotómicas.*** *Sean $n, q \in \mathbb{N}$ tales que $mcd(n, q) = 1$ e $i \in \{0, 1, \ldots, n-1\}$. Se define la i-ésima clase lateral ciclotómica de q módulo n como el conjunto*

$$C_q(i) = \{iq^j \text{ mód } n : j \in \mathbb{N}_0\}$$

siendo el conjunto de las clases ciclotómicas $C_q(i)$ una partición de $\{0, 1, \ldots, n-1\}$.

Ejemplo 22. *Consideremos todas las clases 2-ciclotómicas módulo 7. Como $\text{mcd}(2, 7) = 1$, entonces formarán una partición de $\mathbb{Z}_7$.*

Cuadro 2.2: Clases 2-ciclotómicas módulo 7.

Clase ciclotómica	Elementos de la clase, $C_2(i)$	Clases equivalentes
$C_2(0)$	$\{0\}$	-
$C_2(1)$	$\{1, 2, 4\}$	$C_2(2), C_2(4)$
$C_2(3)$	$\{3, 5, 6\}$	$C_2(5), C_2(6)$

Fuente: Elaboración propia.

Todas las raíces de $x^n - 1$ pueden estudiarse en términos de clases ciclotómicas. A continuación, se detalla el procedimiento [7].

Para factorizar $x^n - 1$ sobre $\mathbb{F}_q$, es conveniente encontrar una extensión $\mathbb{F}_{q^t}$ de $\mathbb{F}_q$ que contenga todas sus raíces, es decir, $\mathbb{F}_q$ debe contener una raíz primitiva n-ésima de la unidad, y para ello es necesario que $n \mid (q^t - 1)$.

Sean $q, n \in \mathbb{N}$ tales que $n \mid (q^t - 1)$ con q potencia de un número primo, y sea γ un elemento primitivo de $\mathbb{F}_{q^s}$ con $s = Ord_n(q)$, es decir, $q^s \equiv 1 \bmod n$. Notemos que $\frac{q^s-1}{n} \in \mathbb{Z}$ y

$$\alpha = \gamma^{\frac{q^s-1}{n}}$$

es una raíz n-ésima de la unidad, esto dado que $\alpha^n = (\gamma^{\frac{q^s-1}{n}})^n = \gamma^{q^s-1} = 1$. Ahora veamos que el orden de α es n, supongamos $0 < r < n$ tal que $\alpha^r = (\gamma^{\frac{q^s-1}{n}})^r = 1$, entonces como $\gamma^{q^s-1} = 1$ se tiene que $q^s - 1 | \frac{(q^s-1)r}{n}$. Es decir, $\frac{(q^s-1)r}{n} = (q^s - 1)m$ para algún $m \in \mathbb{Z}$ por lo tanto tenemos que $\frac{r}{n} = m \in \mathbb{Z}$, lo cual es absurdo. Por tanto, α es una raíz n-ésima de la unidad.

Las raíces de $x^n - 1$ son $1, \alpha, \alpha^2, ..., \alpha^{n-1}$. Entonces, para cada $i \in \{0, 1, \ldots, n-1\}$ como α^i es una raíz se tiene que $\alpha^i, \alpha^{iq}, \alpha^{iq^2} \cdots, \alpha^{iq^{d-1}}$ son raíces del polinomio mónico sobre $\mathbb{F}[x]$ que tiene a α como raíz, y siendo d el menor entero positivo para el cual se tiene $\alpha^{iq^d} = \alpha^i$. De este modo tenemos que el polinomio mínimo para las raíces anteriores está dado por

$$m_{\alpha^i}(x) = (x - \alpha^i)(x - \alpha^{iq}) \ldots (x - \alpha^{iq^{d-1}}),$$

[7] Luis Enrique PINEDA RAMÍREZ. "Códigos BCH y de Reed-Solomon". Trabajo de grado. Benemérita Universidad Autónoma de Puebla, 2020, pág. 89.

donde para el conjunto de exponentes $\{i, iq, iq^2, \ldots, iq^{d-1}\}$, d es el menor entero positivo tal que $iq^d \equiv i (\text{mód} n)$. Estos elementos pueden ser considerados módulo n. En efecto, si $iq^j = kn + r$ para algún $n, k \in \mathbb{Z}$ con $0 \leq r < n$, entonces $\alpha^{iq^j} = \alpha^{kn+r} = \alpha^{kn}\alpha^r = (1)(\alpha^r) = \alpha^r$. Así, podemos considerar el conjunto de exponentes como la clase ciclotómica $C_q(i)$. Por tanto, tenemos

$$m_{\alpha^i} = \prod_{i \in C_q(s)} (x - \alpha^i)$$

y por consiguiente tenemos que $x^n - 1$ es el producto de los distintos polinomios mínimos de sus raíces

$$x^n - 1 = \prod_s M_{\alpha^s},$$

donde s recorre un conjunto de representantes de las clases laterales q-ciclotómicas módulo n.

Sea $T = \cup_s C_s$ la unión de las clases laterales q-ciclotómicas. A las raíces de la unidad $Z = \{\alpha^i | i \in T\}$ se les llama ceros del código cíclico C. El conjunto T es llamado conjunto de definición de C. Nótese que T, y el conjunto de ceros, determinan completamente un polinomio generador $g(x)$, y por tanto la dimensión del código generado por este polinomio $g(x)$ es $n - |T|$, dado que $g(x)$ tiene grado $|T|$.

A continuación se presenta un ejemplo aplicado en SageMath donde haremos uso de la función $codes.CyclicCode(length = n, field = F, D = [\,])$, donde el parámetro $D = [\,]$ corresponde al conjunto de definición del código.

Ejemplo 23. *Calculemos el código cíclico C de longitud 21 sobre $\mathbb{F}_2$ y $D = [1, 3]$ donde estamos tomando como conjunto de definción de C las clases ciclotómicas $C_2(1)$ y $C_2(3)$, adicionalmente hallaremos las clases 2-ciclotómicas módulo 21, para verificar que el conjunto de definición de C es en efecto la colección de clases 2-ciclotómicas módulo n.*

```
sage: F.⟨a⟩ = GF(2)[ ]
sage: n=21
sage: Cc=codes.CyclicCode(length=n, field=F, D=[1, 3])
sage: Cc
```

$[21, 12]\ Cyclic\ Code\ over\ GF(2)$
sage: $Cc.generator\_polynomial()$
$x^9 + x^8 + x^5 + x^4 + x^2 + x + 1$
sage: $Cc.minimum\_distance()$
5
sage: $Cy = Zmod(21).cyclotomic\_cosets(2); Cy$
$[[0], [1, 2, 4, 8, 11, 16], [3, 6, 12], [5, 10, 13, 17, 19, 20], [7, 14], [9, 15, 18]]$
sage: $Cc.defining\_set()$
$[1, 2, 3, 4, 6, 8, 11, 12, 16]$

donde hemos especificado como conjunto definidor las clases laterares 2-ciclótomicas módulo 21, cuyos representantes son la clase $C_2(1)$ y $C_2(3)$, lo cual verificamos con la función de SageMath $Cc.defining\_set()$ que en efecto el cojunto definidor es $C_2(1) \cup C_2(1)$ con $C_2(1) = \{1, 2, 4, 8, 11, 16\}$ y $C_2(3) = \{5, 10, 13, 17, 19, 20\}$, los cuales optenemos con la función $Zmod(21).cyclotomic\_cosets(2)$ que sirve para calcular las clases ciclotómicas.

A continuación se presentan los códigos BCH, los cuales pertenecen a una amplia familia de códigos cíclicos diseñados para corregir errores. Estos códigos representan una generalización de los códigos de Hamming, diseñada específicamente para abordar la corrección de múltiples errores.

2.3. CÓDIGOS BCH

Los códigos BCH representan una extensión significativa de los códigos de Hamming, permitiendo la corrección de múltiples errores. Fueron descubiertos de manera independiente por Bose y Ray-Chaudhuri en 1960, así como por Hocquenghem en 1959. La subclase de los códigos BCH binarios es la subclase más importante desde el punto de vista tanto de la teoría como de la implementación. Una característica importante de los códigos BCH, conocida como la cota BCH, establece que la distancia mínima d es mayor o igual a t. Esta propiedad es particularmente útil ya que proporciona una base sólida para la construcción de códigos, asegurando que cumplan con los requisitos de detección y corrección de errores. Los códigos BCH pueden ser binarios y no binarios, entre los no binarios, los más importantes son los de Reed-Solomon.

Los parámetros de diseño de los códigos BCH se determinan a partir de

ecuaciones específicas que tienen en cuenta el número de errores que se busca corregir. Estas ecuaciones establecen las condiciones necesarias para definir los parámetros adecuados y, por lo tanto, para seleccionar el polinomio generador correcto, los parámetros de diseño estan determinados por:

- Longitud de bloque: $n = q^m - 1$
- Dimensión: $k = n - mt$
- Distancia mínima: $d \geq 2t + 1$

El polinomio generador de este código se especifica en términos de sus raíces sobre $\mathbb{F}_q$.

Definición 2.3.1. *Sea α un elemento primitivo en $\mathbb{F}_{q^m}$, el código BCH de corrección de t errores de longitud $q^m - 1$ y distancia mínima designada $2t + 1$ es un código cíclico tal que su polinomio generador $g(x)$ es el poliomio de menor grado sobre $\mathbb{F}_q$ que tiene a $\alpha, \alpha^2, \alpha^3, \ldots, \alpha^{2t}$ como sus raíces. Sea $\phi_i(x)$ el polinomio minimal de α^i para $1 \leq i \leq 2t$, entonces el polinomio generador está dado por*

$$g(x) = mcm\{\phi_a(x), \phi_{a+1}(x), \ldots, \phi_{a+d-2}(x)\},$$

para algún a, comúnmente se elige $a = 1$, y estos códigos son conocidos como códigos BCH en sentido estricto.

Ejemplo 24. *Supongamos que trabajamos en el cuerpo finito $\mathbb{F}_{2^4}$, lo que significa que $q = 2$ y $m = 4$. Además, consideremos un elemento primitivo $\alpha \in \mathbb{F}_{2^4}$. Construiremos dos códigos BCH de longitud $n = 2^4 - 1 = 15$ y con distancia mínima de diseño $d \geq 2t + 1$, con $t = 2$ y por otro lado con $t = 3$. En primera instancia calcularemos los polinomios minimales $\phi_1, \ldots, \phi_6$ de cada α^i con $0 < i \leq 6$.*

Haciendo uso de la función $.minpoly()$ de SageMath calculamos dichos polinomios

```
sage: F.<a> =GF(2^4, impl = 'ntl')
sage: minpoly(a)
x^4 + x + 1
sage: minpoly(a^2)
x^4 + x + 1
```

sage: $minpoly(a^3)$
$x^4 + x^3 + x^2 + x + 1$
sage: $minpoly(a^4)$
$x^4 + x + 1$
sage: $minpoly(a^5)$
$x^2 + x + 1$
sage: $minpoly(a^6)$
$x^4 + x^3 + x^2 + x + 1$

obteniendo que lo polinomios mínimos de $\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5$ *y* α_6 *son:*

$$\begin{aligned}
\phi_1(x) &= \phi_2(x) = \phi_4(x) = 1 + x + x^4, \\
\phi_3(x) &= \phi_6(x) = 1 + x + x^2 + x^3 + x^4, \\
\phi_5(x) &= 1 + x + x^2.
\end{aligned}$$

Una vez obtenido los polinomios mínimos de α^i *podemos calcular los siguientes códigos:*

1. *El código BCH de doble corrección de errores de longitud* $n = 16 - 1 = 15$*, con dimensión satisfaciendo* $n - k = mt$*, es decir,* $15 - k = 4(2)$ *entonces* $k = 7$ *y distancia mínima de diseño* $d \geq 2t + 1 = 2(2) + 1 = 5$*, es generado por*

   $$g(x) = mcm\{\phi_1(x), \phi_2(x), \phi_3(x), \phi_4(x)\},$$

   luego, dado que $\phi_1(x) = \phi_2(x) = \phi_4(x)$ *y* $\phi_3(x)$ *son polinomios irreducibles distintos, entonces:*

   $$\begin{aligned}
   g(x) &= \phi_1(x)\phi_3(x) \\
   &= (1 + x + x^4)(1 + x + x^2 + x^3 + x^4) \\
   &= 1 + x^4 + x^6 + x^7 + x^8.
   \end{aligned}$$

   El código BCH de doble corrección de errores es el $[15, 7]$ *código cíclico con distancia mínima* $d \geq 5$ *y como el polinomio generador es un polinomio de peso* 5 *entonces la distancia de este código es exactamente* 5*. Verificamos estos parámetros en SageMath a continuación*

sage: $F.\langle x\rangle = GF(2)[\,]$
sage: $C = codes.CyclicCode(1 + x^4 + x^6 + x^7 + x^8, 15)$
sage: C
$[15, 7]$ *Cyclic Code over* $GF(2)$
sage: $C.minimum\_distance()$
5

2. *El código BCH de longitud* 15 *para corrección de* $t = 3$ *errores, es generado por*

$$\begin{aligned} g(x) &= mcm\{\phi_1(x), \phi_2(x), \phi_3(x), \phi_4(x), \phi_5(x), \phi_6(x)\} \\ &= (1 + x + x^4)(1 + x + x^2 + x^3 + x^4)(1 + x + x^2) \\ &= 1 + x + x^2 + x^4 + x^5 + x^8 + x^{10} \end{aligned}$$

generando el $[15, 5]$ *código cíclico con distancia mínima* $d = 7$ *dado que el polinomio generador tiene peso* 7*. Verificamos estos parámetros en SageMath a continuación*

sage: $F.\langle x\rangle = GF(2)[\,]$
sage: $C = codes.CyclicCode(1 + x + x^2 + x^4 + x^5 + x^8 + x^{10}, 15)$
sage: C
$[15, 5]$ *Cyclic Code over* $GF(2)$
sage: $C.minimum\_distance()$
7

En SageMath, podemos verificar que el código de longitud 15 generado por el polinomio $g(x) = 1 + x + x^2 + x^4 + x^5 + x^8 + x^{10}$ corresponde al código BCH con los parámetros establecidos en el ejemplo anterior.

sage: $F.\langle x\rangle = GF(2)[\,]$
sage: $n=15$
sage: $g=1+x^4+x^6+x^7+x^8$
sage: $C1=codes.CyclicCode(generator\_pol=g, length=n)$
sage: $C1$
$[15, 7]$ *Cyclic Code over* $GF(2)$

sage: $C2 = codes.BCHCode(GF(2), 15, 5)$

sage: $C2$

$[15, 7]$ BCH Code over $GF(2)$ with designed distance 5

sage: $C1 == C2$

$True$

3. CÓDIGOS LINEALES COMPLEMENTARIOS DUALES (LCD)

Los códigos LCD fueron introducidos y estudiados por Massey [2]. Estos códigos desempeñan un papel crucial en garantizar la integridad de la información y proteger la seguridad de los datos en diversos entornos de desarrollo. Su aplicación abarca áreas tan variadas como la transmisión de datos en redes de comunicación, la protección de la integridad de datos en sistemas de almacenamiento y una de sus aplicaciones más conocidas: el enmascaramiento de datos.

En entornos donde la seguridad y la confidencialidad son prioritarias, como en el enmascaramiento de datos, los códigos LCD son esenciales. Estos códigos permiten ocultar la información mediante técnicas de codificación, asegurando que los datos confidenciales no sean vulnerables a accesos no autorizados o manipulaciones indebidas, más adelante detallaremos esta importante aplicación.

Definición 3.0.1. ***Código lineal dual complementario (LCD).*** *Un código C es llamado código LCD si $C \cap C^{\perp} = \{0\}$, lo que es equivalente a $C \oplus C^{\perp} = \mathbb{F}_q^n$. Además, observe que si C es un código LCD, entonces $C^{\perp}$ también lo es, dado que $(C^{\perp})^{\perp} = C$.*

Nótese que aunque esta propiedad es inmediata para subespacios vectoriales de característica cero, no es cierto en general para cuerpos de característica prima, como es el caso de los cuerpos finitos.

Ejemplo 25. *Consideremos el cuerpo $\mathbb{F}_2$ el cual tiene característica 2. Y veamos el siguiente código lineal*

sage: $codes.random\_linear\_code(GF(2), 5, 3)$
sage: C
$[5, 3]$ *linear code over* $GF(2)$
sage: $list(C)$
$[(0,0,0,0,0), (0,1,1,0,1), (0,0,1,0,0), (0,1,0,0,1), (1,0,1,0,0), (1,1,0,0,1),$
$(1,0,0,0,0), (1,1,1,0,1)]$
sage: $ct = C.dual\_code()$
sage: ct

$[5, 2]$ *linear code over* $GF(2)$
sage: $list(ct)$
$[(0, 0, 0, 0, 0), (0, 1, 0, 0, 1), (0, 0, 0, 1, 0), (0, 1, 0, 1, 1)]$
sage: $set(C).intersection(set(ct))$
$\{(0, 0, 0, 0, 0), (0, 1, 0, 0, 1)\}$

En este ejemplo obtenemos que $C \cap C^{\perp} = \{(0, 0, 0, 0, 0), (0, 1, 0, 0, 1)\} \neq 0$.

Mas aún, puede ocurrir que uno este contenido en el otro o incluso que sean iguales, lo cual se presenta en la siguiente definición.

Definición 3.0.2. *Sea* C *un* $[n, k]$ *código lineal sobre* $\mathbb{F}_q$

1. Si $C \subseteq C^{\perp}$, entonces C se denomida **auto-ortogonal**.
2. Si $C = C^{\perp}$, entonces C se denomida **auto-dual**.

Nótese que $C^{\perp}$ es un subespacio vectorial de $\mathbb{F}_q^n$, $dim(C^{\perp}) + dim(C) = n$ y $(C^{\perp})^{\perp} = C$. Por lo tanto, si C es un código con parámetros $[n, k]$ entonces su código dual $C^{\perp}$ es un código $[n, n - k]$.

A continuación presentamos un ejemplo de un código dual complementario y mostramos algunas funciones de SageMath para determinar algunas características del código.

Ejemplo 26. *Consideremos el siguiente código generado por la matriz*

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

caculamos en SageMath el código lineal generado por dicha matriz y realizamos otras cuentas como sigue:

sage: $G = Matrix(GF(2), [[1, 0, 0, 0], [1, 1, 0, 0], [1, 1, 1, 0]])$
sage: $C = LinearCode(G); C$
$[4, 3]$ *linear code over* $GF(2)$
sage: $C.is\_self\_orthogonal()$
False

sage: $C.is\_self\_dual()$
False
sage: $Ct = C.dual\_code(); Ct$
$[4, 1]$ *linear code over* $GF(2)$
sage: $A = set(C); B = set(Ct)$
sage: $A.intersection(B)$
$\{(0, 0, 0, 0)\}$
sage: $list(C)$

```
[(0, 0, 0, 0),
 (1, 0, 0, 0),
 (1, 1, 0, 0),
 (0, 1, 0, 0),
 (1, 1, 1, 0),
 (0, 1, 1, 0),
 (0, 0, 1, 0),
 (1, 0, 1, 0)]
```

sage: $list(Ct)$
$[(0, 0, 0, 0), (0, 0, 0, 1)]$
sage: $dimension(C)$
3
sage: $dimension(Ct)$
1

Las funciones de las lineas 3 y 4 nos ayudan a determinar si el código es auto-ortogonal o autodual respectivamente. En este ejemplo, llegamos a la conclusión de que el código no es auto-ortogonal ni autodual. Adicionalmente en la linea 5 obtenemos el $[4, 1]$ *código lineal dual de* C *seguidamente calculamos la intersección de ambos códigos lo que resulta en el conjunto que solo contiene el elemento* $(0, 0, 0, 0)$*. Esto nos lleva a la deducción de que el código es, de hecho, un código dual complementario. Finalmente calculamos las dimensiones de* C *y* $C^{\perp}$ *y podemos notar que en efecto la suma de sus dimensiones es igual a la longitud del código, es decir,* $dim(C^{\perp}) + dim(C) = 1 + 3 = 4$*.*

Nótese también que el hecho de que $dim(C^{\perp}) + dim(C) = n$ no implica que que los códigos sean duales complementarios.

Definición 3.0.3. *Dado* $C \subset \mathbb{F}_q^n$*, decimos que una aplicación* $P : \mathbb{F}_q^n \longrightarrow C$ *es una*

proyección ortogonal sobre C si y solo si

$$P(v) = \begin{cases} v & \textit{si} \quad v \in C \\ 0 & \textit{si} \quad v \in C^{\perp}. \end{cases}$$

Lema 3.0.4. *Sea C un código lineal de longitud n sobre $\mathbb{F}_q$ y sea $P : \mathbb{F}_q^n \longrightarrow \mathbb{F}_q^n$ una aplicación $\mathbb{F}$-lineal. Entonces P es una proyección ortogonal con respecto al producto interno sobre C si y sólo si*

$$P(v) = \begin{cases} v & \textit{si} \quad v \in C \\ 0 & \textit{si} \quad v \in C^{\perp}. \end{cases}$$

Demostración. Supongamos $P : \mathbb{F}_q^n \longrightarrow C$ proyección ortogonal con respecto al producto interno sobre C. Sean $v \in C$ y $u \in C^{\perp}$, entonces tenemos $C = \mathrm{Im}(P)$. Por lo tanto, existe una palabra $x \in \mathbb{F}_q^n$ tal que $P(x) = v$, entonces $v = P(x) = P^2(x) = P(P(x)) = P(v)$, esto es, $P(v) = v \in C$. Por otra parte, como $u \in C^{\perp}$, se tiene que $\langle u, v\rangle = 0$ para todo $v \in C = \mathrm{Im}(P)$. Con lo cual se deduce que $u \in \mathrm{Ker}(P)$, y por lo tanto, $P(u) = 0$. Ahora supongamos que

$$P(v) = \begin{cases} v & \text{si} \quad v \in C \\ 0 & \text{si} \quad v \in C^{\perp}. \end{cases}$$

Como P es función, tenemos que $C \cap C^{\perp} = \{0\}$. Para cada $w \in \mathbb{F}_q^n$, se puede escribir de forma única $w = v + u$, donde $v \in C$ y $u \in C^{\perp}$. Entonces $P(v) = v$ y $P(u) = 0$. Sea $v \in \mathrm{Im}(P)$ y $w \in \mathrm{Ker}(P)$. Entonces $v \in C$ y $P(w) = 0$. Resulta que $w \in C^{\perp}$ y $\langle w, v\rangle = 0$. Por lo tanto, $\mathrm{Im}(P)$ y $\mathrm{Ker}(P)$ son ortogonales con respecto al producto interno. □

Lema 3.0.5. [8] *Sea C un código lineal de longitud n sobre $\mathbb{F}_2^n$ entonces C es un código LCD si, y sólo si la proyección ortogonal sobre C existe.*

Demostración. Supongamos P_C proyección ortogonal sobre C. Entonces

$$P_C(v) = \begin{cases} v & \text{si} \quad v \in C \\ 0 & \text{si} \quad v \in C^{\perp}. \end{cases}$$

Si C no es LCD entonces existe $u \in C \cap C^{\perp}$ con $u \neq 0$, con lo cual tendríamos que $P_C(u) = 0$ y $P_C(u) = 0$, es decir que $0 = u \neq 0$, lo cual es una contradicción.

---

[8] Cristian David Salina Cassiani. “Códigos de grupos LCD y LCP”. 2019.

Ahora, supongamos que C es un código LCD. Sea $w \in \mathbb{F}_q^n$, entonces existen $u \in C$ y $v \in C^\perp$ únicos, tales que $w = u + v$. Consideremos la aplicación $P_C : \mathbb{F}_q^n \longrightarrow \mathbb{F}_q^n$ como $P_C(v) = u$, no es difícil verificar que P_C es una aplicación lineal bien definida. Luego, por el lema anterior tenemos

$$P_C(x) = \begin{cases} x & \text{si} \quad x \in C \\ 0 & \text{si} \quad x \in C^\perp. \end{cases}$$

Ahora bien,

1. veamos que $P_C \in \mathbb{F}_q^n$. Sean $w, w' \in \mathbb{F}_q^n$ tales que $w = u + v$ y $w' = u' + v'$, entonces $P_C(w + w') = P_C[(u + v) + (u' + v'))] = P_C[(u + u') + (v + v'))] = u + u' = P_C(w) + P_C(w')$. Sea α un escalar, $P_C(\alpha w) = P_C(\alpha(u + v)) = P_C(\alpha u + \alpha v) = \alpha u = \alpha P_C(w)$.

2. Sean $C, C' \subset \mathbb{F}_q^n$, tales que $C = C'$, con $P_C(w) = u$ y $P'_C(w) = u$, entonces $u = u$, $P_C(w) = P'_C(w)$ entonces $P_C = P'_C$.
   Por lo tanto, por el lema anterior, P_C es una proyección ortogonal sobre C.

□

Proposición 3.0.6. [8] *Sea C un $[n, k]$ código lineal, y sea G una matriz generadora de C. Entonces C es un código LCD sí, y sólo si la matriz $(k \times k)$ GG^T es no singular.*

Demostración. Supongamos GG^T no singular. Entonces si $u \in C$, es decir que $u = vG$ para algún $v \in \mathbb{F}_q^k$, se tiene

$$\begin{aligned} uG^T(GG^T)^{-1}G &= (vG)G^T(GG^T)^{-1}G \\ &= v(GG^T(GG^T)^{-1}G) \\ &= v(GG^T(G^T)^{-1}G^{-1}G) \\ &= vG \\ &= u. \end{aligned}$$

Además, nótese que si $u \in C^\perp$, es decir $0 = uG^T$. Entonces se tiene

$$uG^T(GG^T)^{-1}G = 0.$$

Con lo cual concluimos que $G^T(GG^T)^{-1}G$ es la proyección ortogonal de C, y por tanto, C es un código LCD.

Ahora, supongamos que GG^T es singular. Como GG^T es una matriz de tamaño $k \times k$, entonces $Rang(GG^T) < k$. Por otro lado, por el teorema del rango-nulidad se tiene que $k = Rank(GG^T) + Null(GG^T)$, donde $Rank(GG^T)$ es el rango de la matriz y $Null(GG^T)$ es la dimensión del espacio nulo (el espacio de los vectores nulos de GG^T). Por tanto tenemos $k = Rank(GG^T) + Null(GG^T) < k + Null(GG^T)$, con lo cual $0 < Null(GG^T)$, lo que implica que $Ker(GG^T) \neq \{0\}$. Por consiguiente, existe un vector $u \in \mathbb{F}_q^k$ distinto de cero tal que $uGG^T = 0$. Nótese que $uG \in C$, con u diferente de cero. Por otra parte, como todo vector $v \in C$ puede escribirse como $v = u'G$ para algún $u' \in \mathbb{F}_q^k$ tenemos

$$\begin{aligned}(uG)v^T &= (uG)(u'G)^T \\ &= uGG^T(u')^T \\ &= 0(u')^T \\ &= 0\end{aligned}$$

por tanto, uG también es un vector en $C^\perp$, es decir, $uG \in C \cap C^\perp \neq \{0\}$, en otras palabras C no es un código LCD. □

A continuación, se presenta un teorema que proporciona una característica fundamental de los códigos LCD cíclicos, estableciendo condiciones bajo las cuales un código cíclico se considera LCD.

3.1. CARACTERIZACIÓN DE LOS CÓDIGOS CÍCLICOS LCD

Antes de presentar el siguiente resultado, es importante comprender el concepto de un polinomio auto-recíproco y de código reversible. El recíproco $p^*(x)$ de un polinomio $p(x)$ de grado n se define por $p^*(x) = x^n p(x^{-1})p_0^{-1}$. Un polinomio se llama auto-recíproco si coincide con su recíproco. Por otro lado, un código es reversible si para cada $(c_0, c_1, \ldots, c_{n-1}) \in C$, la palabra $(c_{n-1}, c_{n-2}, \ldots, c_0) \in C$.

Corolario 3.1.1. *El código cíclico C generado por el polinomio mónico $g(X)$ es reversible si, y sólo si $g(X)$ es auto-recíproco.*

Teorema 3.1.2. *Sea C un código cíclico de longitud n sobre $\mathbb{F}_q$ con polinomio generador $g(x)$, entonces C es un código LCD si cumple una de las siguientes condiciones:*

1. *$g(x)$ es auto-recíproco.*

2. *El inverso de todo cero de $g(x)$, también es un cero de $g(x)$.*

3. $q^l \equiv -1 \bmod n$ *para algún entero positivo l.*

Demostración. Supongamos $C \cap C^{\perp} \neq \{0\}$, entonces existe $p(x) \in \mathbb{F}_q[x]$ no nulo, tal que

$$p(x) = a(x)g(x) \text{ y,}$$
$$p(x) = b(x)g^{\perp}(x)$$

donde $g(x)$ y $g^{\perp}(x)$ son los polinomios generadores de C y $C^{\perp}$, respectivamente.

Por otra parte, como $g(x)$ es auto-recíproco se tiene $g(x) = g^*(x) = x^r g(x^{-1})$ siendo $r \leq n-1$ el grado de $g(x)$, y por otro lado tenemos que $g^{\perp}(x) = \frac{x^k h(x^{-1})}{h_0}$ con $h(x) = \frac{x^n-1}{g(x)}$. Luego,

$$p(x) = a(x)g(x) = a(x)x^r g(x^{-1}) \text{ y,}$$
$$p(x) = b(x)g^{\perp}(x) = b(x)\frac{x^k h(x^{-1})}{h_0} = b(x)\frac{x^k}{h_0}\frac{x^{-n}-1}{g(x^{-1})}$$

entonces, $a(x)x^r g(x^{-1}) = b(x)\frac{x^k}{h_0}\frac{x^{-n}-1}{g(x^{-1})}$. Ahora, si consideramos α raíz de $g(x)$, como $g(x)$ es auto-recíproco tenemos que α^{-1} también es raíz de $g(x)$, por lo que tenemos

$$a(\alpha)\alpha^r g(\alpha^{-1}) = b(\alpha)\frac{\alpha^k}{h_0}\frac{\alpha^{-n}-1}{g(\alpha^{-1})}$$
$$0 = \frac{b(\alpha)\alpha^{k-n}-1}{h_0}\frac{1}{0}$$

lo cual es absurdo. Por tanto, $C \cap C^{\perp} = \{0\}$. □

Nótese que las condiciones anteriores son equivalentes, de la definición de $p^*(x)$ se puede notar que $p^*(x^{-1}) = x^{-n}p(x)$, de lo cual se concluye que β^{-1} es una raíz no nula de $p^*(x)$ si, y solo si β es una raíz no nula de $p(x)$. Por otra parte, sea $C_q(a)$ la clase q-ciclótoma módulo n que contiene a, donde $0 \leq a \leq n-1$, y supongamos $q^l \equiv -1 \bmod n$ para algún entero positivo l. Entonces, $-a \equiv -aq^l \bmod n$. De lo cual concluimos que $-a$ está en $C_q(a)$. Por lo tanto, cada factor irreducible de $x^n - 1$ es auto-recíproco y por tanto reversible.

Ejemplo 27. *Consideremos C el código binario cíclico de longitud 9 con polinomio generador $g(x) = x^6 + x^3 + 1$, con ayuda de SageMath veremos que este código además de ser cíclico, es dual complementario*

```
sage: F.⟨x⟩ = GF(2)[ ]
sage: n = 9
sage: g = x^6 + x^3 + 1
sage: C = codes.CyclicCode(generator_pol = g, length = n)
sage: C
[9, 3] Cyclic Code over GF(2)
sage: C.minimum_distance()
3
sage: Ct = C.dual_code()
sage: Ct
[9, 6] linear code over GF(2)
sage: set(C).intersection(set(Ct))
{(0, 0, 0, 0, 0, 0, 0, 0, 0)}
sage: g.reverse()
x^6 + x^3 + 1
```

Con los resultados obtenidos con cada función en SageMath podemos concluir que el código generado por el polinomio $g(x)$ *es el* $[9, 3, 3]$ *código cíclico y que el código* $C^{\perp}$ *es el* $[9, 6, 2]$ *código cíclico dual, la función* $g.reverse()$ *se usa para verificar si el polinomio es igual a su forma invertida, lo que indica que es un polinomio auto-recíproco, obteniendo que en efecto,* $g(x)$ *es auto-recíproco, además de haberlo comprobado con la intersección de* $C \cap C^{\perp} = \{(0, 0, 0, 0, 0, 0, 0, 0, 0)\}$*, por lo que tenemos que* C *es un código LCD.*

A continuación se presenta un programa desarrollado en SageMath que tiene la capacidad de identificar y presentar una lista de polinomios que cumplen dos propiedades fundamentales: generan códigos cíclicos y son auto-recíprocos.

```
sage: def AllCiclicRec(n, q) :
....:     F.⟨x⟩ = PolynomialRing(GF(q))
....:     D = divisors(x^n − 1)
....:         l = len(D)
....:         for i in range (l) :
....:             if D[i] == D[i].reverse() :
....:             print('Polinomio autor-ecíproco:',D [i])
```

Ejemplo 28. *Haciendo uso del algoritmo anterior obtenemos todos los códigos cíclidos LCD de longitud 7 sobre* $\mathbb{F}_2$

sage: *AllCiclicRec(7,2)*
Factor auto-recíproco = 1
Factor auto-recíproco = $x+1$
Factor auto-recíproco = $x^6+x^5+x^4+x^3+x^2+x+1$
Factor auto-recíproco = x^7+1

Podemos comprobar tomando cualquier polinomio que en efecto los polinomios proporcionados por el programa anterior son auto-recíprocos, por ejemplo, si consideramos el tercer polinomio $g(x) = x^6+x^5+x^4+x^3+x^2+x+1$ *es auto-recíproco*

$$\begin{aligned} x^6 g(x^{-1}) &= x^6(x^{-6}+x^{-5}+x^{-4}+x^{-3}+x^{-2}+x^{-1}+1) \\ &= 1+x+x^2+x^3+x^4+x^5+x^6. \end{aligned}$$

Ejemplo 29. *Empleando el programa anterior obtenemos que los siguientes polinomios son generadores de códigos cíclicos LCD de longitud 3 sobre* $\mathbb{F}_2$*:*

sage: *AllCiclicRec(3,2)*
Factor auto-recíproco = 1
Factor auto-recíproco = $x+1$
Factor auto-recíproco = x^2+x+1
Factor auto-recíproco = x^3+1

ahora, consideremos el polinonio $g(x) = x+1$ *y aplicando las funciones de SageMath que ya hemos visto, comprobamos que en efecto el código generado por este polinomio es un código cíclico LCD, verificando que la intersección del código con su código dual únicamente el elemento* $(0,0,0)$*.*

sage: $F.\langle x\rangle = GF(2)[\,]$
sage: $C{=}codes.CyclicCode(x{+}1,3)$
sage: C
$[3,2]$ $Cyclic\ Code\ over\ GF(2)$
sage: $list(C)$
$[(0,0,0),(1,1,0),(0,1,1),(1,0,1)]$
sage: $Ct{=}C.dual\_code()$ *#Código dual de* C
sage: $list(Ct)$

$[(0,0,0),(1,1,1)]$
sage: $set(C).intersection(set(Ct))$ *# Cálculo de la intersección de C y* $C^{\perp}$
$\{(0,0,0)\}$

A continuación se presenta una aplicación de los códigos lineales duales complementarios.

3.2. ENMASCARAMIENTO DE DATOS

El enmascaramiento de datos es un proceso que implica la modificación de ciertos elementos de los datos almacenados, alterando su contenido mientras se preserva su estructura básica. El objetivo principal es salvaguardar la información confidencial, asegurando que los datos sensibles de los clientes no estén accesibles fuera del entorno de producción. Mediante el enmascaramiento de datos, creamos una versión de los datos que conserva una estructura similar a la original, pero que no contiene información auténtica. Esta técnica resulta especialmente útil en entornos de desarrollo o pruebas de software. El propósito fundamental es proteger la integridad de los datos reales mientras se dispone de un sustituto funcional para situaciones en las que no se requieren los datos originales. Es una técnica que consiste en combinar los datos sensibles con valores aleatorios llamados máscaras, de tal manera que la información original se oculte y sea difícil de obtener sin conocer las máscaras correspondientes. A continuación se presenta el esquema de enmascaramiento propuesto por J. Bringer [5].

Definición 3.2.1. ***Complemento de un espacio vectorial.*** *El espacio vectorial* C *se puede completar con algunos vectores para generar* $\mathbb{F}_2^n$*. Estos vectores definen el complemento* D *de* C *en* $\mathbb{F}_2^n$*, escribimos* $\mathbb{F}_2^n = C \oplus D$ *para decir que* $\mathbb{F}_2^n$ *es la suma directa de* C *y* D*.*

Si C es un código lineal, donde G y H son matices generadoras de C y D (complemento de C) respectivamente, entonces todo vector $z \in \mathbb{F}_q^n$ se puede escribir de forma única como

$$z = xG + yH, \tag{3.1}$$

para dado $x \in \mathbb{F}_q^k$ y dado $y \in \mathbb{F}_q^{n-k}$.

Si C es un código lineal dual complementario ($D = C^{\perp}$), entonces $\mathbb{F}_2^n = C \oplus C^{\perp}$, y además tenemos que $GH^T = 0$ (la matriz de ceros $k \times (n-k)$), donde H es la matriz de control del código lineal C. Entonces, de la ecuación 3.1 x y y pueden ser recuperados de la siguente forma:

$$x = zG^T(GG^T)^{-1}, \tag{3.2}$$

$$y = zH^T(HH^T)^{-1}. \tag{3.3}$$

Nótese que dado que G es una base para C, es decir, está compuesta de vectores linealmente independientes de $\mathbb{F}_2^n$, entonces GG^T es una matriz invertible $k \times k$, así mismo tenemos que HH^T es una matriz invertible $(n-k) \times (n-k)$. Por lo que la proyección P_C y P_D de $z \in \mathbb{F}_2^n$ en C y en D, está dada por:

$$P_C : \mathbb{F}_2^n \longrightarrow C, \quad z \longmapsto c = zG^T(GG^T)^{-1}G,$$

$$P_D : \mathbb{F}_2^n \longrightarrow D = C^{\perp}, \quad z \longmapsto d = zH^T(HH^T)^{-1}H.$$

Representación de datos. Para el esquema de enmascaramiento, elegimos C y D, como $D = C^{\perp}$. Representaremos cualquier vector $z \in \mathbb{F}_2^n$ como la suma de dos palabras de código $z = c \oplus d$. Los datos sensibles codificados son $c \in C$, mientras que la máscara es $d \in D$.

Entonces, para proteger datos confidenciales x de k bits, se requieren $(n-k)$ bits aleatorios. Estos se denotan por y, la máscara es igual a $d = yH$. La idea es que la información sean palabras en clave y que las máscaras actúen como ruido añadido intencionalmente. Pero, como la información y la máscara viven en dos subespacios suplementarios, siempre es posible recuperar ambos, utilizando las ecuaciones 3.2 y 3.3.

Ejemplo 30. *Supongamos que una empresa está desarrollando un nuevo sistema de gestión de datos y que para ello se requieren realizar pruebas de software, durante las cuales es necesario proporcionar ciertos datos, como el número de identificación de los usuarios (número de cédula). Con el fin de proteger la confidencialidad y la integridad de este dato durante su procesamiento, la empresa empleará técnicas de enmascaramiento de datos. El siguiente programa en SageMath permite llevar a cabo este proceso.*

Supongamos que el número de cédula de un usuario es $123.456.789$*. El siguiente*

programa permite enmascarar este dato una vez convertido en su representación binaria:

```
sage: def DataMask(Cédula) :
....:     F.<a> = GF(2)[ ]
....:     z = [ ]
....:     B = bin(Cédula)[2 :]
....:     P = vector([int(num) for num in B])
....:     C = codes.CyclicCode(x + 1, length = 7)
....:     G = C.generator_matrix()
....:     H = C.parity_check_matrix()
....:     k = C.dimension()
....:     S = [P[i : i + k] for i in range(0, len(P), k)]
....:     l = len(S)
....:     v = random_vector(GF(2), len(C[0]) - k)
....:         for i in range(0,l-1):
....:             z1 = S[i] * G + v * H
....:             z.append(z1)
....:     print('Dimensión=',k)
....:     print('Cédula en binario=',P)
....:     print('S=',S)
....:     print('Datos Enmascarados=',z)
sage: DataMask(12345678)
Dimensión= 6
Cédula en binario= (1,0,1,1,1,1,0,0,0,1,1,0,0,0,0,1,0,1,0,0,1,1,1,0)
S=[(1, 0, 1, 1, 1, 1), (0, 0, 0, 1, 1, 0), (0, 0, 0, 1, 0, 1), (0, 0, 1, 1, 1, 0)]
Datos Enmascarados = [(0, 0, 0, 1, 1, 1, 0), (1, 1, 1, 0, 1, 0, 1), (1, 1, 1, 0, 0, 0, 0),
(1, 1, 0, 1, 1, 0, 1)]
```

La asignación de S *lo que permite es dividir el vector inical* P *que corresponde al dato cédula*$= 123.456.789$ *en binario, es decir*

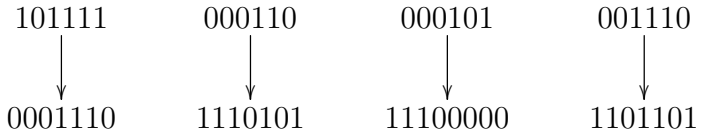
$$P = (1,0,1,1,1,1,0,0,0,1,1,0,0,0,0,1,0,1,0,0,1,1,1,0)$$

en vectores cuya longitud es la dimensión del código, es decir de longitud $k = 6$ *en este caso, esto dado que los datos sensibles son datos en* $\mathbb{F}_q^k$*, obteniendo el*

conjunto de datos sensibles

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

El programa reasigna nuevos valores a estos datos



y estos datos son los que serán usados en la prueba de software, empleando como información el dato $00011101110101111000001101101$ *correspondiente al número* $62.582.893$ *en el sistema decimal, en lugar de,* 101111000110000101001110 *correspondiente al dato real* $123.456.789$*, permitiendo que los desarrolladores y probadores trabajen con datos simulados que son funcionalmente equivalentes pero no revelan información real de los clientes.*

De esta manera, las empresas se aseguran de no exponer ni comprometer datos sensibles de sus usuarios mientras realizan pruebas críticas de software. El enmascaramiento de datos se convierte en una herramienta esencial para proteger la privacidad y la confidencialidad de los clientes, al tiempo que permite pruebas eficientes y seguros.

4. CÓDIGOS LINEALES LOCALMENTE RECUPERABLES (LRC)

Los códigos localmente recuperables surgieron en respuesta al creciente uso de técnicas de codificación en sistemas de almacenamiento distribuido o almacenamiento en la nube. En un contexto de sistemas de almacenamiento fiable y eficiente, se busca la capacidad de recuperar información de un nodo en caso de que falle o no esté disponible, aprovechando la información almacenada en otros nodos. Los códigos localmente recuperables, como su nombre lo indica, permiten la reparación de datos codificados que han sido perdidos mediante un procedimiento local, es decir, usando una pequeña cantidad de datos en lugar de hacer uso de toda la información contenida en la palabra código. En este sentido establecemos la siguiente definición.

4.1. RECUPERACIÓN LOCAL DE COORDENADAS

En un sistema de almacenamiento distribuido, los datos se dividen y almacenan en múltiples servidores, cada uno con su propio procesador y un disco duro de capacidad limitada. Su estructura presenta dos desafíos principales:

1. Fallos en los discos duros. Los discos duros tienen una vida útil de aproximadamente 3 años, siendo propensos a fallos, lo que podría resultar en una pérdida permanente de datos, lo cual es inaceptable en un entorno de almacenamiento de datos críticos.

2. La posibilidad de que un servidor deje de responder. Ya sea debido a la sobrecarga, actualizaciones o reinicios, puede causar retrasos en la atención de las solicitudes de los usuarios, lo cual es tolerable hasta cierto punto, pero indeseable.

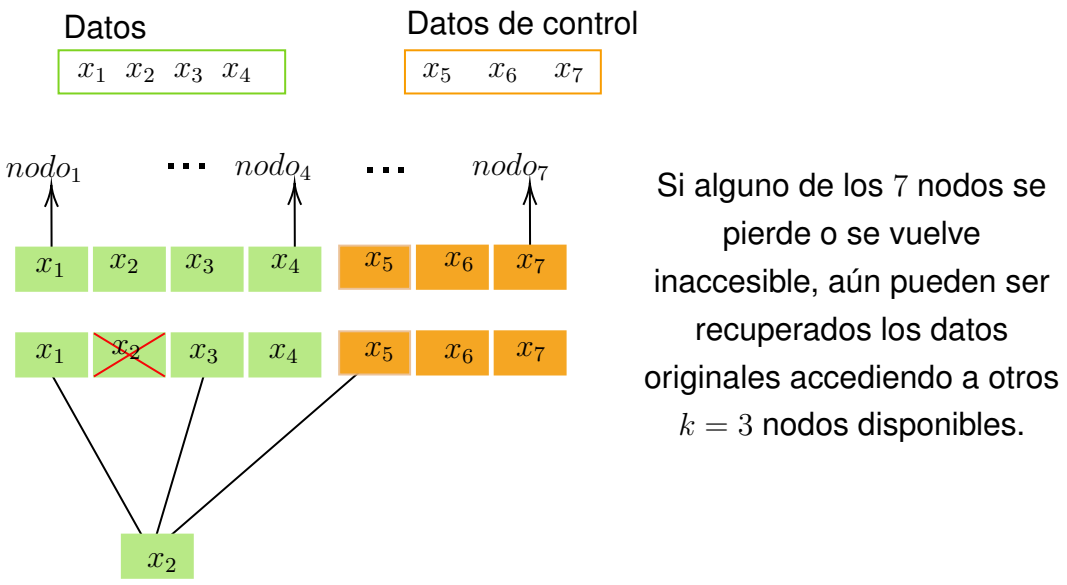
A lo largo del tiempo, en desarrollo del almacenamiento distribuido, se han implementado tres enfoques clave para abordar estos desafíos:

1. **Replicación triple:** En las etapas iniciales del almacenamiento distribuido, la estrategia consistía en triplicar todos los datos. En términos de confiabililidad es lo suficientemente bueno ya que aborda los problemas de fallo de discos duros y servidores. Sin embargo, es altamente ineficiente en cuanto a términos de costo de almacenamiento, ya que triplica los recursos.

2. **Códigos correctores de errores:** en la siguiente etapa se introdujeron los códigos correctores de errores como los códigos de Reed-Solomon.

   Ejemplo 31. *El* $[7, 3, 5]$ *código Reed-Solomon, el cual utiliza* 4 *simbolos originales y agrega* 3 *simbolos de conttrol. Donde la longitud del código corresponde al número de nodos que emplearemos en el almacenamiento. La distancia mínima de este código es* $d = 2(n - k) + 1 = 4 + 1 = 5$*, por tanto, este código tiene la capacidad de corregir hasta* 2 *errores.*

Figura 4.1: Esquema de recuperación de un nodo perdido en el $[7, 3, 5]$ código Reed-Solomon.



Fuente: Elaboración propia.

De modo que este código proporciona igual confiabilidad que la triplicación de datos pero un costo de almacenamiento menor, ya que con el $[7, 3, 5]$ código Reed-Solomon solo se necesita almacenar una cantidad adicional de datos $(3$ datos de control$)$ que equivale a $1, 5$ veces la cantidad de datos originales. Esto significa que el costo total de almacenamiento es solo un $1, 5$ veces mayor en lugar de tres veces, lo que lo hace más eficiente en términos de costo.

La razón por la que no se usan códigos más largos, como el $[14, 10]$ código de Reed-Solomon es porque aunque se proporcionaría mayor confiabilidad y menor costo de almacenamiento, también requeriría acceder a un mayor

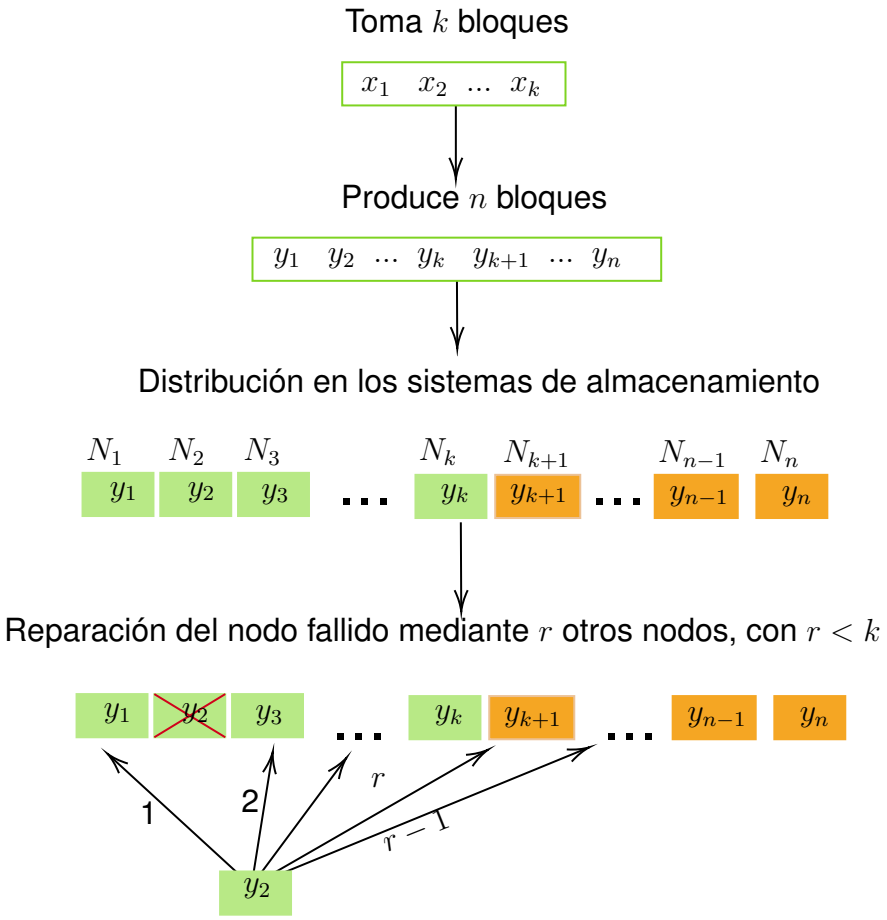
número de servidores para recuperar los datos originales, lo que llevaría más tiempo y recursos.

3. **Códigos localmente recuperables:** la tercera generación busca la combinación ideal de confiabilidad, eficiencia y recuperación rápida. Los códigos localmente recuperables son la respuesta a esta necesidad. Estos códigos pueden proteger contra un gran número de borrados y, al mismo tiempo, ofrecen un algoritmo de recuperación local rápido para abordar un número limitado de borrados. Además, minimizan la redundancia de almacenamiento, lo que los hace altamente eficientes y efectivos en entornos de almacenamiento distribuido.

En la teoría de la codificación, los códigos localmente recuperables (LRC) han emergido como un tema sumamente atractivo, dada su relevancia en sistemas de almacenamiento distribuido a gran escala, donde la pérdida de un solo nodo de almacenamiento se considera un evento de error bastante común. En un mundo donde la disponibilidad y la integridad de los datos son importantes y necesarios, los LRC han demostrado ser una herramienta valiosa. Estos códigos ofrecen una capacidad única para recuperar datos incluso cuando un nodo de almacenamiento falla, y lo hacen de manera local, sin necesidad de recurrir a complejos procesos.

Imagine un inmenso sistema de almacenamiento distribuido, con una red de servidores interconectados. En este entorno, es casi inevitable que algunos nodos experimenten fallos o pérdida de datos. Aquí es donde los códigos LRC brillan con su capacidad para recuperar información de manera eficiente, sin la necesidad de replicar excesivamente los datos o recurrir a operaciones costosas de recuperación. En lugar de considerar un solo fallo como un enorme problema, los LRC permiten que estos sistemas sigan funcionando con relativa normalidad, minimizando el impacto de las fallas en la experiencia del usuario. Esta flexibilidad es especialmente valiosa en aplicaciones de almacenamiento en la nube y otros entornos distribuidos, donde la confiabilidad y la eficiencia son fundamentales. El siguiente esquema ilustra de cierto modo este proceso.

Figura 4.2: Esquema general de recuperación local de un nodo perdido en sistemas de almacenamiento distribuido.



Fuente: Elaboración propia.

Veamos la siguiente definición, la cual se refiere a la localidad de un código.

Definición 4.1.1. *Un código $C \subseteq \mathbb{F}_q$ es LRC con localidad r, si para cada coordenada $i \in [n]$ de una palabra código de C puede ser recuperada por a lo sumo r valores de otras coordenadas del código. Es decir, si existe un subconjunto $R_i \subset [n] \setminus i$ con $|R_i| \leq r$ y una función recuperadora $\phi : {\mathbb{F}_q}^r \longrightarrow \mathbb{F}_q$ tal que para cada palabra código $c \in C$ tenemos*

$$c_i = \phi(c_{j_1}, \cdots, c_{j_r}) = \phi(\{c_j, j \in R_i\}) \tag{4.1}$$

$$c_i = \alpha_{j_1} c_{j_1 + \ldots} + \alpha_{j_r} c_{j_r}$$

donde $j_k \in R$ y $j_1 < j_2 < \cdots < j_r$.

Usaremos la notación $[n, k, r]$ para hacer referencia a un código LRC con cardinalidad q^k, localidad r y longitud n, y además tenemos que la distancia mínima de C satisface la cota como singleton

$$d \leq n - k - \left\lceil \frac{k}{r} \right\rceil + 2. \tag{4.2}$$

Para su demostración recomendamos ver [9], pág.6027.

Los códigos que alcanzan la igualdad en esta cota se conocen como códigos LRC óptimos.

Ejemplo 1. *Haremos uso del* $[7, 4]$ *código binario de Hamming para verificar que la distancia mínima del código satisface 4.2.*

Como ya mostramos anteriormente, este código de Hamming está dado por palabras de siete símbolos $a, b, c, d, e, f, g \in \{0, 1\}$. Donde a, b, c, d son elegidos arbitrariamente y e, f, g son los simbolos de paridad. Y se tiene

$$a + b + d + e = 0$$
$$a + c + d + f = 0$$
$$b + c + d + g = 0$$

Ahora, veamos que para cada coordenada de una palabra del código puede ser recuperada por a lo más 3 otras coordenadas del código.

Supongamos que al momento de transmitir una palabra del código (a, b, c, d, e, f, g) se pierde la información de la coordenada 0, es decir, se pierde o modifica el valor de a. Para poder recuperar ese valor, podemos hacer uso de las coordenadas $2, 4, 5$ o de $4, 5, 6$. La siguiente tabla muestra los conjuntos recuperadores para cada coordenada y la cantidad total de otras coordenadas necesarias para recuperar

---

[9] *et al.* GOPALAN Parikshit. "On the Locality of Codeword Symbols". En: *IEEE Transactions on Information Theory* 58.11 (2012), págs. 6925-6934. DOI: 10.1109/TIT.2012.2208937.

Cuadro 4.1: Conjuntos recuperadores de la i-ésima coordenada.

Coordenada perdida	Conjuntos Recuperadores Coordenadas recuperadoras	Total coordenas mínimas necesarias
0	$\{1,3,4\}$, $\{2,3,5\}$, $\{1,5,6\}$, $\{2,4,6\}$, $\{1,2,3,4,5,6\}$	3
1	$\{0,3,4\}$, $\{2,3,6\}$, $\{1,5,6\}$, $\{2,4,5\}$, $\{0,2,3,4,5,6\}$	3
2	$\{0,3,5\}$, $\{1,3,6\}$, $\{0,4,6\}$, $\{1,4,5\}$, $\{0,1,3,4,5,6\}$	3
3	$\{0,1,4\}$, $\{0,2,5\}$, $\{1,2,6\}$, $\{4,5,6\}$, $\{0,1,2,4,5,6\}$	3
4	$\{0,1,3\}$, $\{3,5,6\}$, $\{1,2,5\}$, $\{0,2,6\}$, $\{0,1,2,3,5,6\}$	3
5	$\{0,2,3\}$, $\{3,4,6\}$, $\{0,1,4\}$, $\{1,2,5\}$, $\{0,1,2,3,4,6\}$	3
6	$\{1,2,3\}$, $\{3,4,5\}$, $\{0,1,5\}$, $\{0,2,4\}$, $\{0,1,2,3,4,5\}$	3

Fuente: Elaboración propia.

Por lo que tenemos el $[7,4,3]$ código de Hamming con localidad $r = 3$ y la distancia mínima satisface la cota 4.2

$$d \leq n - k - \left\lceil \frac{k}{r} \right\rceil + 2 \text{ se tiene } 3 \leq 7 - 4 - \left\lceil \frac{4}{3} \right\rceil + 2 = 4$$

Nota 2. *Un código MDS de parámetros $[n,k,d]$ verifica que su localidad es k.*

Definición 4.1.2. *Códigos LRC para múltiples borrados: se dice que el código C tiene localidad (r,δ) ($[n,k,r,\delta]$ código LRC), $\delta > 1$, si para cada coordenada $i \in [n]$ existe un conjunto $R_i \subset [n]$ tal que $|R_i| \leq r+\delta-1$, $i \in R_i$ y la distancia mínima de la restricción del código $C_{R_i} = \{(c_{j_1},\ldots,c_{j|R_i|})|j_1,\ldots,j_{|R_i|} \in R_i$ y $a(c_1,\ldots,c_n) \in C\}$ es al menos δ.*

En un $[n,k,r,\delta]$ código LRC la distancia mínima d satisface la siguiente cota

$$d \leq n - k - \left\lceil \frac{k}{r} \right\rceil (\delta - 1) + \delta \tag{4.3}$$

Para su demostración recomendamos ver [10], pág.2777.

Notemos que para $\delta = 2$ esta definión se reduce a la definición anterior.

---

[10] *et al.* PRAKASH Neetha. "Optimal Linear Codes with a Local-Error-Correction Property". En: *IEEE International Symposium on Information Theory - Proceedings* (feb. de 2012). DOI: 10.1109/ISIT.2012.6284028.

Ejemplo 32. *En el ejemplo 1 donde la distancia mínima del código es* $d = 3$ *y localidad* $r = 3$ *tenemos:*

$$\begin{aligned} d &\leq n - k - \left\lceil \frac{k}{r} \right\rceil (\delta - 1) + \delta \\ 3 &\leq 7 - 4 - \left\lceil \frac{4}{3} \right\rceil (\delta - 1) + \delta \\ 3 &\leq 3 - 2\delta + 2 + \delta \\ 0 &\leq -\delta + 2 \\ \delta &\leq 2. \end{aligned}$$

Por lo que la posibilidad para delta es $\delta = 1$ *ó* $\delta = 2$*, pero como* $\delta > 1$*, entonces se tiene que* δ *debería ser* 2*. Ahora, estudiando cada coordenada que conforma una palabra del código consideremos el primer conjunto recuperador de cada coordenada* i*-ésima del Cuadro 4.1 y agregando dicha coordenada al conjunto, tenemos:*

Cuadro 4.2: Conjuntos restringidos para la i-ésima coordenada.

Coordenada i-ésima	Conjunto R_i
0	$R_0 = \{0, 1, 3, 4\}$
1	$R_1 = \{0, 1, 3, 4\}$
2	$R_2 = \{0, 2, 3, 5\}$
3	$R_3 = \{0, 1, 3, 4\}$
4	$R_4 = \{0, 1, 3, 4\}$
5	$R_5 = \{0, 2, 3, 5\}$
6	$R_6 = \{1, 2, 3, 6\}$

Fuente: Elaboración propia.

Con lo cual vamos a considerar los códigos restringidos en los conjuntos de coordenadas: $R_0 = R_1 = R_3 = R_4$*,* $R_2 = R_5$ *y* R_6*. A continuación se presenta un programa diseñado en SageMath que nos permite obtener estos códigos restringidos y su respectiva distancia mínima:*

```
sage: def CodRestringido(V) :
....:     F = GF(2)
....:     H = matrix(GF(2), [[1,0,1,1,0,1,0], [1,1,0,1,1,0,0], [0,1,1,1,0,0,1]])
```

```
. . . :   C = codes.from_parity_check_matrix(H)
. . . :   s = len(V))
. . . :       for k in range(0,s):
. . . :         CR = [[C[i][v[k][0]], C[i][v[k][1]], C[i][v[k][2]], C[i][v[k][3]]] for i in
. . . :         range(len(C))]
. . . :         print('Código restringido:',CR)
. . . :         d=DistanciaMin(CR)
. . . :         print('Distancia mínima:',d)
```

donde H es una matriz de control del $[7, 4, 3]$ código de Hamming que estamos empleando, CR es la lista de vectores con cooredenadas em R_i y la función $DistanciaMin(CR)$ es la siguiente función que permite calcular la distancia mínima de los vectores que conforman cada código restringido.

```
sage: def DistanciaMin(C) :
. . . :   l = C
. . . :   L = l[0]
. . . :   d = [ ]
. . . :       for i in range (0, len(l)) :
. . . :           distancia=sum ([l[i][j]! = 0for j in range(L)])
. . . :            if distancia! = 0 :
. . . :              d= d.append(d)
. . . :       d = min(d)
. . . :       return min(d)
```

A continuación, ejecutamos la función ***CodRestringido(V)*** *donde V es una lista de vectores que indican las cooredenadas a las cuales se va a restringrir el código,*

```
sage: CodRestringido([vector([0, 1, 3, 4]), vector([0, 2, 3, 5]), vector([1, 2, 3, 6])])
Código restringido: [[0, 0, 0, 0], [1, 0, 0, 1], [0, 1, 0, 1], [1, 1, 0, 0], [0, 0, 0, 0], [1, 0, 0, 1],
[0, 1, 0, 1], [1, 1, 0, 0], [0, 0, 1, 1], [1, 0, 1, 0], [0, 1, 1, 0], [1, 1, 1, 1], [0, 0, 1, 1], [1, 0, 1, 0],
[0, 1, 1, 0], [1, 1, 1, 1]]
Distancia mínima 2
Código restringido: [[0, 0, 0, 0], [1, 0, 0, 1], [0, 0, 0, 0], [1, 0, 0, 1], [0, 1, 0, 1], [1, 1, 0, 0],
```

$[0,1,0,1],[1,1,0,0],[0,0,1,1],[1,0,1,0],[0,0,1,1],[1,0,1,0],[0,1,1,0],[1,1,1,1],$
$[0,1,1,0],[1,1,1,1]]$
Distancia mínima 2
Código restringido: $[0,0,0,0],[0,0,0,0],[1,0,0,1],[1,0,0,1],[0,1,0,1],[0,1,0,1],$
$[1,1,0,0],[1,1,0,0],[0,0,1,1],[0,0,1,1],[1,0,1,0],[1,0,1,0],[0,1,1,0],[0,1,1,0],$
$[1,1,1,1],[1,1,1,1]]$
Distancia mínima 2

Donde la distancia de cada código restringido en R_i *es* 2*, por tanto hemos comprobado que para cada coordenada* $i \in [7]$ *existe un conjunto* $R_i \subset [7]$ *con* $|R_i| = 3 \leq r + \delta - 1 = 3 + 2 - 1 = 4$ *y la distancia de cada código* C_{R_i} *es* $d = 2 \geq \delta = 2$*, y que satisface la Cota 4.3*

$$3 \leq 7 - 4 - \left\lceil \frac{4}{3} \right\rceil (2-1) + 2 = 3.$$

Por tanto, el $[7,4]$ *código binario de Hamming es un* $[7,4,3,2]$ *código LRC, con localidad* $(3,2)$*.*

Definición 4.1.3. *Disponibilidad de un código. Se dice que un código tiene disponibilidad* (r,t) *si para coordenada* $c_i, i \in [n]$ *de un* $[n,k,d]$ *código, si existen* t *subconjuntos* $R_l(i) \subset [n]$ *para* $l = 1,2,\ldots,t$*, tales que*

1. $i \in P_l(i)$ *y* $|P_l(i)| \leq r+1$.
2. $P_l(i) \cap P_{l'}(i) = i, 1 \leq l \neq l' \leq t$.
3. c_i *es una función de los símbolos indexados por* $P_l(i) \setminus \{i\}$*.*

La disponibilidad de un código es de suma importancia en sistemas de almacenamiento distribuido. Dado que se refiere a la capacidad del código para recuperar la información de un símbolo específico a partir de otros símbolos, especialmente en situaciones donde algunos nodos pueden fallar o volverse inaccesibles.

La disponibilidad se convierte en un componente esencial para garantizar la integridad de la información en entornos distribuidos. Por ejemplo, si un nodo falla y existen nodos adicionales capaces de recuperar la información del nodo perdido, la disponibilidad se manifiesta al acceder a esos nodos recuperadores. Es crucial considerar la situación en la que, aunque uno de los

nodos recuperadores pueda estar experimentando fallas o inaccesibilidad, la disponibilidad puede persistir si otros nodos recuperadores aún están operativos y pueden ser utilizados para recuperar la información del nodo perdido.

Ejemplo 33. *Continuando con el ejemplo anterior, dado el cuadro 4.1, que contiene los conjuntos de coordenadas necesarios para obtener la i-ésima coordenada perdida del código, se observa que no hay subconjuntos recuperadores disjuntos dos a dos. Por lo tanto, la disponibilidad del código LRC* $[7, 4, 3, 2]$ *es* $t = 1$.

Esto implica que, en caso de que se pierda el nodo que almacena un símbolo específico del código, y si uno de los nodos que almacena los datos de un conjunto recuperador presenta fallos o simplemente no está disponible, no habría forma de recuperar la información perdida. La disponibilidad de $t = 1$ *indica que, al menos, un conjunto recuperador completo debe estar completamente disponible y funcional para recuperar la información perdida en cualquier circunstancia.*

4.2. CARACTERIZACIÓN DE LOS CÓDIGOS CÍCLICOS LRC

A continuación se presenta un resultado que nos permite asociar los conjuntos recuperadores de un código C dual complementario con respecto a su código dual $C^{\perp}$, y más aún, con respecto al código $\overleftarrow{C^{\perp}}$ el cual fue definido en 2.1.8.

Teorema 4.2.1. *Sea C un código lineal dual complementario y $C^{\perp}$ el código dual de C, entonces cada $a(x) \in C^{\perp}$ de peso $r + 1$ define un grupo recuperador para C al igual que el código invertido $\overleftarrow{C^{\perp}}$.*

Demostración. Sean $c = (c_0, c_1, \ldots, c_{n-1}) \in C$ y $a = (a_0, a_1, \ldots, a_{n-1}) \in C^{\perp}$, entonces, por definición tenemos

$$c \cdot a = \sum_{i=0}^{n-1} c_i \cdot a_i = 0$$

dado que a tiene $r + 1$ coordenadas no nulas, es decir $\omega(a) = r + 1$, al conjunto de coordenadas donde a es distinto de cero se le denomina $supp(a)$, es decir $supp(a) = \{j_0, j_1, \ldots, j_r | a_{j_m} \neq 0, m \in \{0, 1, \ldots, r\}\}$, entonces tenemos

$$c \cdot a = \sum_{j \in supp} c_j \cdot a_j = 0$$

luego, para cualquier $k \in J \subset \{0, \ldots, n-1\}$ tenemos

$$c_k = -a_k^{-1} \sum_{\substack{j \in supp(a) \\ j \neq k}} c_j \cdot a_j$$

dado que $\sum_{\substack{j \in supp(a) \\ j \neq k}} c_j \cdot a_j = c_{j_0} a_{j_0} + c_{j_1} a_{j_1} + \ldots + c_{j_r} a_{j_r} = 0$ y por consiguiente

$$c_{j_0} a_{j_0} = -c_{j_1} a_{j_1} - \ldots - c_{j_r} a_{j_r} = - \sum_{\substack{j \in supp(a) \\ j \neq k}} c_j \cdot a_j$$

$$c_{j_0} = -(a_{j_0})^{-1} \sum_{\substack{j \in supp(a) \\ j \neq k}} c_j \cdot a_j.$$

Con lo cual, se concluye que las coordenadas no nulas de una palabra código $a \in C^{\perp}$ constituyen un grupo recuperador para el código C, puesto que cualquier coordenada perdida de $c \in C$ puede ser recuperada por r otras coordenadas. Además, note que si $a \in C^{\perp}$ tiene coordenadas con peso $\omega(a) = r + 1$, entonces $\overleftarrow{a} \in \overleftarrow{C^{\perp}}$ también, y por tanto, define un grupo recuperador de C. □

En [11] se han estudiado los códigos cíclicos LRC con localidad r, y presentan el siguiente resultado.

Teorema 4.2.2. *Sea $(r+1)|n$, $r|k$, $n|(q-1)$. Sea $\alpha \in \mathbb{F}_q$ una raíz primitiva n-ésima de la unidad, y sea C un $[n, k]$ código cíclico con ceros $\alpha^i, i \in \mathcal{Z} = \mathcal{L} \cup \mathcal{D}$, donde*

$$\mathcal{L} = \{1 + l(r+1) | l = 0, \ldots, \frac{n}{r+1} - 1\}$$
$$\mathcal{D} = \{1, \ldots, n - \frac{k}{r}(r+1) + 1\}.$$

Entonces C es un $[n, k, r]$ código LRC.

Esta construcción puede ser extendida a códigos con localidad (r, δ). Para esto, asumimos que $r|k$, tomamos $m = r + \delta - 1$, y toma los ceros del código como

$$\mathcal{L} = \left\{ i + lm \mid l = 0, 1, \ldots, \frac{n}{m} - 1, i = 1, 2, \cdots \delta - 1 \right\} \subseteq Z$$
$$\mathcal{D} = \{1, 2, \ldots, n - k - ((k/r) - 1)(\delta - 1)\}.$$

---

[11] CHEN, Zitan y BARG, Alexander. "Cyclic and Convolutional Codes With Locality". En: *IEEE Transactions on Information Theory* 67.2 (2021), págs. 755-769. DOI: 10.1109/TIT.2020.3031207.

El siguiente algoritmo permite obtener un código cíclico que satisface las condiciones del teorema anterior y tiene como conjunto de definición $\mathcal{Z} = \mathcal{L} \cup \mathcal{D}$.

```
sage: def LRC_r(n, q, k, D1):
....:     F.<x>=PolynomialRing(GF(q))
....:     D=divisors(x^n - 1)
....:     l = len(D)
....:         for i in range (0, l):
....:             C=codes.CyclicCode(generator_pol=D[i], length=n)
....:             k1=C.dimension()
....:             k1=C.dimension()
....:             Df=C.defining_set()
....:             if k1==k and Df==D1:
....:                 g=C.generator_polynomial()
....:                 print(C)
....:                 print(g)
....:                 print(Df)
```

Una vez definidos los parámetros del código, utilizamos el algoritmo $LRC\_r(n, q, k, D1)$, donde para $D1$ ingresamos el conjunto de definición deseado para el código.

Ejemplo 34. *Sean $q = 29$, $n = 14$ y $r + 1 = 7$, nótese que $n|q - 1$ y $(r + 1)|n$. Por otro lado, elijamos $k = 12$, con $(r = 6)|(12 = k)$. Luego, hallaremos un $[14, 12]$ código cíclico con conjunto de definición $\mathcal{Z} = \mathcal{L} \cup \mathcal{D}$, donde*

$$\mathcal{L} = \{1 + l(r + 1)|l = 0, \ldots, \frac{n}{r + 1} - 1\} = \{1 + 7l|l = 0, 1\} = \{1, 8\}$$
$$\mathcal{D} = \{1, \ldots, n - \frac{k}{r}(r + 1) + 1\} = \{1\}.$$

Calculamos $LRC_r(14, 29, 12, [1, 8])$ y obtenemos

```
sage: LRC_r(14, 29, 12, [1, 8])
[14, 12] Cyclic Code over GF(29)
x^2 + 13
[1, 8]
```

Obteniendo de este modo el $[14, 12, 6]$ código LRC.

5. CÓDIGOS CÍCLICOS LRC-LCD

Como hemos visto a lo largo de este trabajo los códigos LRC y LCD tienen una cantidad enorme de ventajas sobre problemas de almacenamiento distribuido y el salvaguardado de datos sensibles. Charul Rajput y Maheshanand Bhaintwal en [12] establecieron una conexión entre los códigos LRC y los códigos LCD, y mostraron algunas condiciones para la construcción de códigos cíclicos LRC-LCD con localidad (r, δ), proporcionando códigos óptimos útiles en los sistemas de almacenamiento de datos distribuidos.

5.1. CARACTERIZACIÓN DE CÓDIGOS CÍCLICOS LRC-LCD CON LOCALIDAD (r, δ)

Sea $n|(q-1)$, $m = r + \delta - 1$ tal que $m|n$, y sea α una raíz enésima de unidad en $\mathbb{F}_q$. Sea C un código cíclico de longitud n sobre $\mathbb{F}_q$ con conjunto de ceros Z. Si

$$L = \left\{ i + lm \mid l = 0, 1, \ldots, \frac{n}{m} - 1, i = 1, 2, \cdots \delta - 1 \right\} \subseteq Z$$

entonces C es un (r, δ) código LRC.

Teorema 5.1.1. *Sea q una potencia prima, $n|(q-1)$ y α una raíz enésima de unidad en $\mathbb{F}_q$. Sea k, r y δ números enteros positivos tales que $1 \leq r \leq k$, δ es par, $m|n$ y $\left(\frac{nr}{m} - k\right)$ es par, donde $m = r + \delta - 1$. Sea*

$$L = \left\{ i + j \mid -\left(\frac{\delta - 2}{2}\right) \leq i \leq \left(\frac{\delta - 2}{2}\right), j \equiv 0 \text{ mód } m \right\}$$

y

$$D = \{ s \mid s = -t, -t+1, \ldots, -1, 0, 1, \cdots, t-1, t \}$$

donde t se define de la siguiente manera:

1. Si $c = \frac{1}{2}\left(\frac{n}{m} - \frac{k}{r}\right)$ es un entero, entonces t está en el rango

$$cm - \left(\frac{\delta - 2}{2}\right) \leq t \leq cm + \left(\frac{\delta - 2}{2}\right)$$

---

[12] RAJPUT, Charul y BAINTWAL, Maheshanand. "Cyclic LRC-LCD Codes With Hierarchical Locality". En: *IEEE Communications Letters* 25.3 (2021), págs. 711-715. DOI: 10.1109/LCOMM.2020.3040170.

2. Si $c = \frac{1}{2}\left(\frac{n}{m} - \frac{k}{r}\right)$ no es un número entero, entonces t satisface

$$t - (\delta - 1)\left\lfloor \frac{t}{m} \right\rfloor = cr + \frac{\delta - 2}{2}$$

y se encuentra en el rango $cm - \frac{\delta}{2} - \frac{\delta-1}{r} < t < cm + \frac{\delta-2}{2}$.

Si C es un código cíclico de longitud n, con conjuntos de ceros $Z = L \cup D$, entonces C es un código LRC$-$LCD de dimensión k y localidad (r, δ). La distancia mínima d de C satisface el límite inferior

$$d \geq n - k - \left\lceil \frac{(k+s)(\delta-1)}{r} \right\rceil - \delta + 2. \tag{5.1}$$

Además, si se cumple el caso 1, C es un código LRC óptimo.
La sigueinte desmostración de este teorema fue realizada por Charul Rajput y Maheshanand Bhaintwal en [12] pág. 713.

Demostración. Primero veamos que C es un código LCD. Sea $u \in Z = L \cup D$; si $u \in D$ entonces es claro que $-u \in D$. Si $u \in L$ entonces $u = i + j$ con $-\left(\frac{\delta-2}{2}\right) \leq i \leq \left(\frac{\delta-2}{2}\right)$ y $j = 0$ mód m. Por lo tanto $n - u = n - (i + j) = (-i) + (n - j) \in L$, como $n - j = 0$ mód m entonces por teorema 3.1.2, C es un código LCD.

El conjunto L garantiza que la localidad del código C es (r, δ). Ahora la dimensión de C es $k = n - |Z| = n - (\mid L \mid + \mid D \mid - \mid L \cap D \mid)$, donde $\mid L \mid = \frac{n}{m}(\delta - 1)$ y $\mid D \mid = 2t + 1$. Sea $t = t'm + a$, donde t' y a son enteros con $0 \leq a \leq m - 1$. La intersección de los conjuntos L y D depende del valor de t y puede determinarse en los dos casos siguientes:

1) Suponga $c = \frac{1}{2}\left(\frac{n}{m} - \frac{k}{r}\right)$ es un entero. Entonces tenemos los siguientes dos casos:

- Si $cm \leq t \leq cm + \left(\frac{\delta-2}{2}\right)$ entonces $t' = c$ y $0 \leq a \leq \frac{\delta-2}{2}$. Por lo tanto, $|L \cap D| = 2\left(\left\lfloor \frac{t}{m} \right\rfloor (\delta - 1) + a\right) + 1 = 2(c(\delta - 1) + a) + 1$, y

$$\begin{aligned} |Z| &= \frac{n}{m}(\delta - 1) + 2t + 1 - (2(c(\delta - 1) + a) + 1) \\ &= \frac{n}{m}(\delta - 1) + 2(cm + a - c(\delta - 1) - a) \\ &= \frac{n}{m}(\delta - 1) + 2cr. \end{aligned}$$

- Si $cm - \left(\frac{\delta-2}{2}\right) \leq t < cm$ entonces $t' = c - 1$ y $m - \frac{\delta-2}{2} \leq a \leq m - 1$. Por lo tanto, $|L \cap D| = 2\left(\left\lfloor \frac{t}{m} \right\rfloor + 1\right)(\delta - 1) - m + a) + 1 = 2(c(\delta - 1) - m + a) + 1$, y

$$\begin{aligned} |Z| =& \frac{n}{m}(\delta - 1) + 2t + 1 \\ & - (2(c(\delta - 1) - m + a) + 1) \\ =& \frac{n}{m}(\delta - 1) + 2((c - 1)m + a - c(\delta - 1) - a) \\ =& \frac{n}{m}(\delta - 1) + 2cr. \end{aligned}$$

2) Ahora supongamos $c = \frac{1}{2}\left(\frac{n}{m} - \frac{k}{r}\right)$ no es un entero. Entonces tenemos $t - (\delta - 1)t' = cr + \frac{\delta-2}{2}$. Sustituyendo $t' = \frac{t-a}{m}$, obtenemos

$$t = \left(c + \frac{\delta - 2}{2r}\right) m - \left(\frac{\delta - 1}{r}\right) a.$$

Ya que $cm - \frac{\delta}{2} - \frac{\delta-1}{r} < t < cm + \frac{\delta-2}{2}$, de (3), obtenemos

$$\frac{\delta - 2}{2} < a < m - \left(\frac{\delta - 2}{2}\right).$$

Por lo tanto, $|L \cap D| = 2\left(\left\lfloor \frac{t}{m} \right\rfloor (\delta - 1) + \frac{\delta-2}{2}\right) + 1$, y

$$\begin{aligned} |Z| =& \frac{n}{m}(\delta - 1) + 2t + 1 \\ & - \left(2\left(\left\lfloor \frac{t}{m} \right\rfloor (\delta - 1) + \frac{\delta - 2}{2}\right) + 1\right) \\ =& \frac{n}{m}(\delta - 1) + 2\left(t - \left\lfloor \frac{t}{m} \right\rfloor (\delta - 1) - \left(\frac{\delta - 2}{2}\right)\right). \\ =& \frac{n}{m}(\delta - 1) + 2cr, \text{ dado que } \left\lfloor \frac{t}{m} \right\rfloor = t' \end{aligned}$$

Por lo tanto la dimensión de C es $n - |Z| = n - \left(\frac{n}{m}(\delta - 1) + 2cr\right) = n\left(\frac{m-\delta+1}{r}\right) - 2r\left(\frac{1}{2}\left(\frac{n}{m} - \frac{k}{r}\right)\right) = k$.

Ahora, el límite inferior en la distancia mínima de C es determinado para los casos 1 y 2, como sigue:

1) Para el caso $1, t$ toma el valor en el rango $\mathrm{cm} - \left(\frac{\delta-2}{2}\right) \leq t \leq cm + \left(\frac{\delta-2}{2}\right)$. Ahora para cualquier entero a,

$$\left\{ i \mid am - \left(\frac{\delta-2}{2}\right) \leq i \leq am + \left(\frac{\delta-2}{2}\right) \right\} \subseteq L.$$

Por tanto, $Z = L \cup D$ contiene

$2\left(cm + \frac{\delta-2}{2}\right) + 1$ raíces consecutivas

$\left\{ i \mid -\left(cm + \left(\frac{\delta-2}{2}\right)\right) \leq i \leq cm + \left(\frac{\delta-2}{2}\right) \right\}$ de C.

Por tanto, la distancia mínima de C satisface

$$\begin{aligned} d &\geq 2\left(cm + \frac{\delta-2}{2}\right) + 2 \\ &= 2m\left(\frac{1}{2}\left(\frac{n}{m} - \frac{k}{r}\right)\right) + (\delta - 2) + 2 \\ &= n - \frac{km}{r} + \delta = n - k - \frac{k}{r}(\delta - 1) + \delta. \end{aligned}$$

Esta distancia alcanza el límite 5.1. Por lo tanto C es óptimo.

2) Para el caso 2,observamos que D contiene $2t + 1$ ceros consecutivos de C. Así que la distancia mínima de C satisface

$$\begin{aligned} d &\geq 2t + 2 \\ &> 2\left(cm - \frac{\delta}{2} - \frac{\delta-1}{r}\right) + 2 \\ &= 2m\left(\frac{1}{2}\left(\frac{n}{m} - \frac{k}{r}\right)\right) - \delta - \frac{2(\delta-1)}{r} + 2 \\ &= n - \frac{k}{r}(r + \delta - 1) - \delta - \frac{2(\delta-1)}{r} + 2 \\ &= n - k - \frac{(k+2)(\delta-1)}{r} - \delta + 2. \end{aligned}$$

□

Ejemplo 35. *Sean $q = 37$, $n = 36$, $r = 3$ y $\delta = 4$. Entonces, $m = r + \delta - 1 = 6$ y por consiguiente*

$$L = \left\{ i + j \mid -\left(\frac{\delta-2}{2}\right) \leq i \leq \left(\frac{\delta-2}{2}\right), j \equiv 0 \bmod m \right\} = \{ i + j \mid -1 \leq i \leq 1, j \equiv 0 \bmod m \}$$
$$= \{-13, -12, -11, -7, -6, -5, -1, 0, 1, 5, 6, 7, 11, 12, 13, 17, 18, 19\}.$$

Ahora, para calcular el conjunto D, consideremos $r \leq k = 16$ entonces $c = \frac{1}{2}\left(\frac{36}{6} - \frac{16}{3}\right) = \frac{1}{3}$, lo cual no es un entero, por tanto tenemos que t se define a

partir del caso 2*. Es decir,*

$$t-(\delta-1)\left\lfloor\frac{t}{m}\right\rfloor=cr+\frac{\delta-2}{2}\Rightarrow t-3\cdot\left\lfloor\frac{t}{6}\right\rfloor=\frac{1}{2}\cdot 3+\frac{2}{2}\Rightarrow t-3\cdot\left\lfloor\frac{t}{6}\right\rfloor=2$$

y se encuentra en el rango

$$cm-\frac{\delta}{2}-\frac{\delta-1}{r}<t<cm+\frac{\delta-2}{2}\Rightarrow\frac{1}{3}\cdot 6-\frac{4}{2}-\frac{\delta-1}{3}<t<\frac{1}{2}\cdot 6+\frac{2}{2}\Rightarrow -1<t<3$$

lo que implica que t *debe ser* 2 *y por tanto*

$$D=\{s\mid s=-t,-t+1,\ldots,-1,0,1,\cdots,t-1,t\}=\{-2,-1,0,1,2\}$$

y por tanto, la distancia mínima de C *satisface* $d\geq 6$*.*

Calculamos en SageMath el código cíclico de longitud 36 *sobre* $\mathbb{F}_{37}$ *cuyo conjunto de definición es*

$$\mathcal{Z}=\mathcal{L}\cup\mathcal{D}=[-13,-12,-11,-7,-6,-5,-2,-1,0,1,2,5,6,7,11,12,13,17,18,19]$$

verificaremos que en efecto es un código cíclico con dimensión $k=36$ *y comprobaremos que es un código LCD.*

sage: $F=GF(37,'a')$
sage: $n=36$
sage: $Z=[-13,-12,-11,-7,-6,-5,-2,-1,0,1,2,5,6,7,11,12,13,17,18,19]$
sage: $Cc=codes.CyclicCode(length=n,field=F,D=Z)$
sage: Cc
$[36,16]\ Cyclic\ Code\ over\ GF(37)$
sage: $g=Cc.generator\_polynomial();\ g$
$x^{20}+5x^{19}+x^{18}+35x^{14}+27x^{13}+35x^{12}+2x^8+10x^7+2x^6+36x^2+32x+36$

hemos comprobado que el código Cc *bajo el conjunto de definición construido anteriormente tiene dimensión* 16*, además no es dificil verificar que el polinomio generador* g *de este código es auto-recíproco calculando* $g_0^{-1}g(x^{-1})x^{20}=36^{-1}g(x^{-1})x^{20}=36g(x^{-1})x^{20}\ \text{mód}\ 37$ *y como resultado debemos obtener* $g(x)$*, lo cual haremos en el siguiente algoritmo.*

```
sage: def AuReci(P, q) :
...:     import sympy
...:     x=sympy.symbols('x')
...:     c=sympy.Poly(P).all_coeffs()
...:     r=sympy.Poly(P).degree(x)
...:     Q=x^-1
...:     ev=P.subs(x, Q)
...:     P1=x^r
...:     final=sympy.expand(ev * P1)
...:     co=sympy.Poly(final).all_coeffs()
...:     inv=sympy.mod_inverse(co[0], q)
...:     res=[(inv * i) %q for i in co ]
...:     pol=sympy.Poly.from_list(res, x)
...:     return(res==c)
```

Este programa consiste en comparar los coeficientes del polinomio $P(x)$ con los coeficientes del polinomio $P_0^{-1}P(x^{-1})x^r \mod q$ donde r es el grado del polinomio P. Hemos importado la librería para matemáticas símbolicas para crear variables y manipular expresiones algebráicas. Por otro lado, la función $sympy.Poly(P).all\_coeffs()$ genera la lista de los coeficientes de P, $c = sympy.Poly(P).degree(x)$ es el grado del polinomio, $ev = P.subs(x, Q)$ lo es la función que evalúa x^{-1} en $P(x)$, la función $final = sympy.expand(ev \cdot P1)$ es el producto de $x^r \cdot P(x^{-1})$ y para concluir, la función $inv = [(inv * i) \,\%q$ for i in co $]$ es utilizada para obtener hacer al polinomio mónico. Finalmente lo que hace el programa es comparar los coeficientes de ambos polinomios y si coinciden es porque el polinomio es auto-recíproco.

Haciendo uso de este algoritmo verificamos que el polinomio generador del código obtenido en 35 es auto-recíroco, y por tanto es LCD

```
sage: AuReci(x^20 + 5*x^19 + x^18 + 35*x^14 + 27*x^13 + 35*x^12 + 2*x^8 + 10*x^7 + 2*x^6 +
36*x^2 + 32*x + 36, 37)
True
```

Concluyendo así, que hemos construido un $[36, 16]$ código cíclico LRC-LCD con localidad $(3, 4)$.

A continuación, se presenta un algoritmo diseñado en SageMath, que resulta altamente útil al permitirnos evitar cálculos manuales y construir códigos cíclicos LRC-LCD. Este algoritmo computa eficientemente el conjunto de ceros $\mathcal{Z} = \mathcal{L} \cup \mathcal{D}$, tal como se define en el teorema 5.1.1, para un código con parámetros específicos q, n, r, y δ. Estos parámetros deben satisfacer las condiciones establecidas en el mismo teorema.

```
sage: def Zeros(n, q, k, δ) :
...:     F.⟨a⟩ = GF(2)[ ]
...:     m=r + s − 1
...:     a=(s − 2)/2
...:     L=[ ]
...:     I=range(−a, a + 1)
...:     J=range(0, n, m)
...:     for i in I :
...:         for j in : J
...:         f = (i + j) %n
...:         L.append(f)
...:     print('ConjuntoL :', L)
...:     c=(1/2)(n/m − k/r)
...:     if (c − int(c))==0 :
...:             t=cm − (s − 2)/2
...:             if t==0 :
...:                 t=cm + (s − 2)/2
...:     else:
...:      i=1
...:      si=True
...:      while  si:
...:        t1=cr + (s − 2)/2
...:        t=i − (s − 1)int((i/m))
...:        i=i + 1
...:        if  t==t1 :
...:            si=false
...:     D=range(−t, t + 1)
...:    print('ConjuntoD :', list(D))
...:    A=set(L)
```

```
...: B=set(L)
...: In=A.intersection(B)
...: A1 = list(In)
...: L1=L + list(D)
...: Z1=[i for i in L1 if i not in A1]
...: Z=Z1 + A1
...: print('Ceros del código:', Z)
...: C=codes.CyclicCode(length=n, fiel= = F, D=Z)
...: g=C.generator_polynomial()
...: print(C)
...: print(g)
```

Ejemplo 36. *Con ayuda del algoritmo anterior vamos a construir un código LRC con parámetros* $n = 36$*,* $q = 37$*,* $k = 16$*,* $r = 3$ *y* $\delta = 4$

sage: $def\ Zeros((36, 37, 16, 3, 4)$:
Conjunto L: $[35, 5, 11, 17, 23, 29, 0, 6, 12, 18, 24, 30, 1, 7, 13, 19, 25, 31]$
Conjunto D: $[-2, -1, 0, 1, 2]$
Ceros del código: $[35, 5, 11, 17, 23, 29, 6, 12, 18, 24, 30, 7, 13, 19, 25, 31, -2, -1, 2, 0, 1]$
$[36, 16]$ *Cyclic Code over GF(37)*
$x^{20} + 5x^{19} + x^{18} + 35x^{14} + 27x^{13} + 35x^{12} + 2x^8 + 10x^7 + 2x^6 + 36x^2 + 32x + 36$

y con ayuda del algoritmo $AuReci$ *verificamos que este código es LCD*

sage: $AuReci(x^{20} + 5x^{19} + x^{18} + 35x^{14} + 27x^{13} + 35x^{12} + 2x^8 + 10x^7 + 2x^6 + 36x^2 + 32x + 36, 37)$
$True$

De este modo concluimos que se trata de un $[36, 16]$ *código cíclico LRC-LCD con localidad* $(3, 4)$*.*

6. CONCLUSIONES

A lo largo de este trabajo, se abordaron con éxito problemas críticos en el ámbito de la teoría de las comunicaciones, tales como el problema almacenamiento distribuido y la protección de la integridad de los datos sensibles. La investigación detallada de la familia de códigos cíclicos desempeñó un papel crucial en la solución de estos desafíos. En particular, el esquema de enmascaramiento definido sobre códigos LCD es una respuesta al desafío de proteger la integridad y confidencialidad de los datos sensibles. Además, se enfatizó la importancia de los códigos localmente recuperables (LRC), los cuales emergieron como un tema de gran atractivo. Su relevancia en sistemas de almacenamiento distribuido a gran escala radica en la capacidad para abordar la pérdida de nodos de almacenamiento, un evento de error considerado bastante común en dichos sistemas.

El estudio detallado de las propiedades de la familia de códigos cíclicos permitió la construcción de códigos cíclicos LRC-LCD. Este enfoque combinado de códigos cíclicos LRC y LCD demostró ser una estrategia prometedora y efectiva para resolver problemas fundamentales en almacenamiento distribuido y protección de datos sensibles.

La combinación de códigos cíclicos LRC-LCD son una respuesta a dos de los problemas modernos de la teoría de la información, el problema de confidencialidad de datos sensibles y el problema de almacenamiento distribuido o almacenamiento en la nube.

Cabe destacar que el empleo de SageMath resultó de gran ayuda y utilidad para verificar resultados relevantes y facilitar la construcción de códigos con parámetros específicos de interés. La aplicación de esta herramienta contribuyó significativamente a la validación y comprensión de los resultados obtenidos a lo largo de la investigación.

BIBLIOGRAFÍA

BRINGER Julien, *et al.* “Orthogonal Direct Sum Masking”. En: (jun. de 2014), págs. 40-56. DOI: 10.1007/978-3-662-43826-8_4 (vid. págs. 12, 65).

CHEN, Zitan y BARG, Alexander. “Cyclic and Convolutional Codes With Locality”. En: *IEEE Transactions on Information Theory* 67.2 (2021), págs. 755-769. DOI: 10.1109/TIT.2020.3031207 (vid. pág. 79).

CLAUDE, Carlet y SYLVAIN, Guilley. “Complementary dual codes for counter-measures to side-channel attacks”. En: *Advances in Mathematics of Communications* 10.1 (2016), págs. 131-150. DOI: 10.3934/amc.2016.10.131 (vid. pág. 11).

DIMITRIS, Papailiopoulos y ALEXANDROS, Dimakis. “Locally Repairable Codes”. En: *IEEE Transactions on Information Theory* 60.10 (2014), págs. 5843-5855. DOI: 10.1109/TIT.2014.2325570 (vid. pág. 11).

GOPALAN Parikshit, *et al.* “On the Locality of Codeword Symbols”. En: *IEEE Transactions on Information Theory* 58.11 (2012), págs. 6925-6934. DOI: 10.1109/TIT.2012.2208937 (vid. pág. 73).

HUFFMAN, Cary y PLESS, Vera. *Fundamentals of Error-Correcting Codes*. Cambridge University Press, 2003. DOI: 10.1017/CBO9780511807077.

JAMES, Massey. “Linear codes with complementary duals”. En: *Discrete Mathematics* 106-107 (1992), págs. 337-342. DOI: https://doi.org/10.1016/0012-365X(92)90563-U (vid. págs. 11, 56).

MUNUERA, Carlos y TENÓRIO, Wanderson. “Locally recoverable codes from rational maps”. En: *Finite Fields and Their Applications* 54 (2018), págs. 80-100. DOI: https://doi.org/10.1016/j.ffa.2018.07.005.

PINEDA RAMÍREZ, Luis Enrique. “Códigos BCH y de Reed-Solomon”. Trabajo de grado. Benemérita Universidad Autónoma de Puebla, 2020, pág. 89 (vid. pág. 49).

PRAKASH Neetha, *et al.* “Optimal Linear Codes with a Local-Error-Correction Property”. En: *IEEE International Symposium on Information Theory - Proceedings* (feb. de 2012). DOI: 10.1109/ISIT.2012.6284028 (vid. pág. 74).

RAJPUT, Charul y BAINTWAL, Maheshanand. “Cyclic LRC-LCD Codes With Hierarchical Locality”. En: *IEEE Communications Letters* 25.3 (2021), págs. 711-715. DOI: 10.1109/LCOMM.2020.3040170 (vid. págs. 81, 82).

Salina Cassiani, Cristian David. “Códigos de grupos LCD y LCP”. 2019 (vid. págs. 59, 60).

SENDRIER, Nicolas. “Linear codes with complementary duals meet the Gilbert–Varshamov bound”. En: *Discrete Mathematics* 285.1 (2004), págs. 345-347. DOI: https://doi.org/10.1016/j.disc.2004.05.005.

YANG, Xiang y MASSEY, James. “The condition for a cyclic code to have a complementary dual”. En: *Discrete Mathematics* 126.1 (1994), págs. 391-393. DOI: https://doi.org/10.1016/0012-365X(94)90283-6 (vid. pág. 11).

ZHU Bing, *et al.* “An Improved Bound and Singleton-Optimal Constructions of Fractional Repetition Codes”. En: *IEEE Transactions on Communications* 70.2 (2022), págs. 749-758. DOI: 10.1109/TCOMM.2021.3128543.