Algoritmo de Sugiyama para códigos skew RS

Doble Grado en Ingeniería Informática y Matemáticas

Guillermo Galindo Ortuño

Trabajo Fin de Grado
E.T.S. de Ingenierías Informática y de Telecomunicación
Facultad de Ciencias

17 de septiembre de 2020

Índice

Introducción

Conceptos básicos de la teoría de la comunicación

Un código permite añadir información adicional a un mensaje para su transmisión a través de un canal.

El algoritmo de Sugiyama para códigos skew RS permite corregir errores presentes en un mensaje codificado tras su transmisión por un canal ruidoso.

Conceptos básicos de la teoría de la comunicación

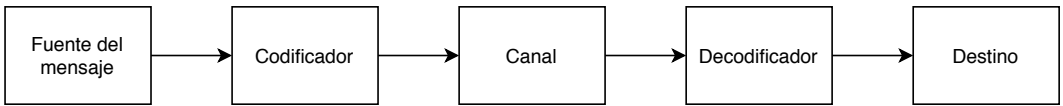

Figura: modelo de comunicación de Shannon

Conceptos básicos de la teoría de la comunicación

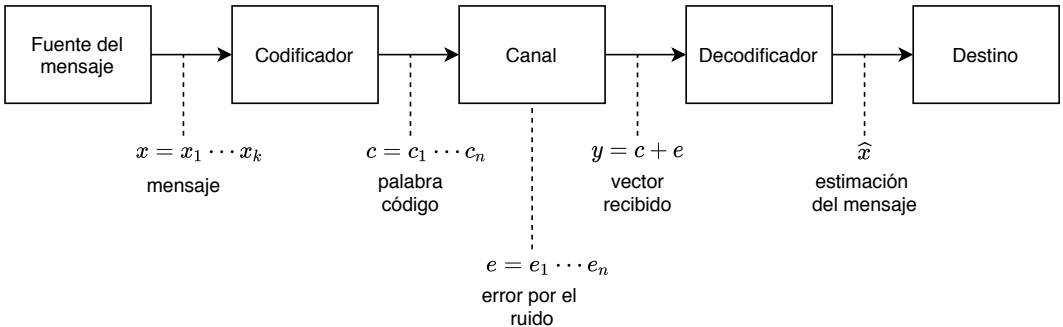

Figura: modelo de comunicación de Shannon

Objetivos

1. Estudiar y presentar el algoritmo de Sugiyama para códigos skew RS.

Objetivos

1. Estudiar y presentar el algoritmo de Sugiyama para códigos skew RS.
   - Estudiar conceptos básicos de la teoría de códigos.

Objetivos

1. Estudiar y presentar el algoritmo de Sugiyama para códigos skew RS.
   - Estudiar conceptos básicos de la teoría de códigos.
   - Estudiar las extensiones de Ore.

Objetivos

1. Estudiar y presentar el algoritmo de Sugiyama para códigos skew RS.
   - Estudiar conceptos básicos de la teoría de códigos.
   - Estudiar las extensiones de Ore.
   - Estudiar los códigos cíclicos sesgados.

Objetivos

1. Estudiar y presentar el algoritmo de Sugiyama para códigos skew RS.
   - Estudiar conceptos básicos de la teoría de códigos.
   - Estudiar las extensiones de Ore.
   - Estudiar los códigos cíclicos sesgados.
   - Presentar el algoritmo y demostrar su funcionamiento.

Objetivos

1. Estudiar y presentar el algoritmo de Sugiyama para códigos skew RS.
   - Estudiar conceptos básicos de la teoría de códigos.
   - Estudiar las extensiones de Ore.
   - Estudiar los códigos cíclicos sesgados.
   - Presentar el algoritmo y demostrar su funcionamiento.
2. Realizar una implementación en el entorno SageMath de dicho algoritmo, junto con la estructura de clases necesaria para representar la familia de códigos cíclicos sesgados.

Teoría de Códigos

Conceptos básicos

Definición (Código lineal)

Un $[n, k]$ código lineal $\mathcal{C}$ de longitud n y dimensión k sobre un cuerpo $\mathbb{F}$ es un subespacio vectorial de $\mathbb{F}^n$ de dimensión k.

Conceptos básicos

Definición (Código lineal)

Un $[n, k]$ código lineal $\mathcal{C}$ de longitud n y dimensión k sobre un cuerpo $\mathbb{F}$ es un subespacio vectorial de $\mathbb{F}^n$ de dimensión k.

Definición (Matriz generadora)

Sea $\mathcal{C}$ un $[n, k]$ código lineal sobre $\mathbb{F}$, diremos que G de dimensiones $k \times n$ es una matriz generadora de $\mathcal{C}$ si y solo si sus filas forman una base de $\mathcal{C}$.

Conceptos básicos

Definición (Código lineal)

Un $[n, k]$ código lineal $\mathcal{C}$ de longitud n y dimensión k sobre un cuerpo $\mathbb{F}$ es un subespacio vectorial de $\mathbb{F}^n$ de dimensión k.

Definición (Matriz generadora)

Sea $\mathcal{C}$ un $[n, k]$ código lineal sobre $\mathbb{F}$, diremos que G de dimensiones $k \times n$ es una matriz generadora de $\mathcal{C}$ si y solo si sus filas forman una base de $\mathcal{C}$.

Definición (Distancia de Hamming)

Dados dos vectores $x, y \in \mathbb{F}^n$, se define la distancia de Hamming entre ellos $d(x, y)$ como el número de coordenadas en que difieren.

Codificar

Sea $\mathcal{C}$ un $[n, k]$ código lineal sobre $\mathbb{F}$, y G una matriz generadora de $\mathcal{C}$, existe una biyección entre sus elementos y los elementos de $\mathbb{F}^k$.

Por esto, el método más comun para codificar un mensaje es verlo como un elemento de $\mathbb{F}^k$, digamos $x = x_1 \cdots x_k$, y codificarlo como la palabra código

$$c = xG.$$

Decodificar

Supongamos que recibimos un vector $y = c + e \in \mathbb{F}^n$, donde e es el error producido en la transmisión del mensaje por un canal ruidoso.

Decodificar consiste en determinar la palabra c (y por tanto el mensaje x) al recibir un vector y.

Decodificar

El método de decodificación más usado es el *vecino más cercano*, y consiste en encontrar la palabra código más cercana al mensaje recibido.

Decodificar

El método de decodificación más usado es el *vecino más cercano*, y consiste en encontrar la palabra código más cercana al mensaje recibido.

Los algoritmos de decodificación más simples calculan de forma iterativa la distancia del mensaje recibido a cada una de las palabras código. Esto es muy ineficiente.

Decodificar

El método de decodificación más usado es el *vecino más cercano*, y consiste en encontrar la palabra código más cercana al mensaje recibido.

Los algoritmos de decodificación más simples calculan de forma iterativa la distancia del mensaje recibido a cada una de las palabras código. Esto es muy ineficiente.

Un resultado importante sobre este método de decodificación es que si d es la distancia mínima entre palabras de un código, se pueden corregir un máximo de $\lfloor (d-1)/2 \rfloor$ errores.

Códigos cíclicos

Definición (Código cíclico)

Diremos que un código lineal $\mathcal{C}$ de dimensión n es cíclico si y solo si, para toda palabra código $c = c_0 c_1 \cdots c_{n-1}$, el vector $c_{n-1} c_0 \cdots c_{n-2}$ también está en $\mathcal{C}$.

Códigos cíclicos

Definición (Código cíclico)

Diremos que un código lineal $\mathcal{C}$ de dimensión n es cíclico si y solo si, para toda palabra código $c = c_0 c_1 \cdots c_{n-1}$, el vector $c_{n-1} c_0 \cdots c_{n-2}$ también está en $\mathcal{C}$.

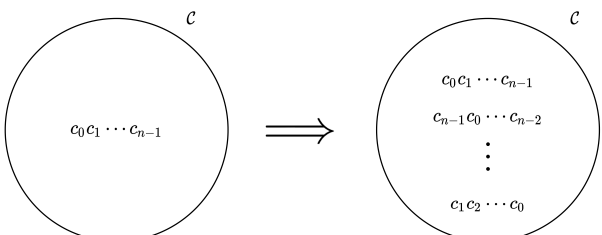

Figura: condición de ciclicidad

Códigos cíclicos

Una forma de representar las palabras código es utilizando su forma polinomial.

$$c = c_0 c_1 \cdots c_{n-1} \in \mathbb{F}^n \longleftrightarrow c(x) = c_0 + c_1 x + \cdots + c_{n-1} x^{n-1} \in \mathbb{F}[x].$$

Códigos cíclicos

Una forma de representar las palabras código es utilizando su forma polinomial.

$$c = c_0 c_1 \cdots c_{n-1} \in \mathbb{F}^n \longleftrightarrow c(x) = c_0 + c_1 x + \cdots + c_{n-1} x^{n-1} \in \mathbb{F}[x].$$

Utilizando esta correspondencia, la condición de ciclicidad es equivalente la siguiente condición:

$$c(x) \in \mathcal{C} \implies xc(x) \bmod x^n - 1 \in \mathcal{C}.$$

Códigos cíclicos

Una forma de representar las palabras código es utilizando su forma polinomial.

$$c = c_0 c_1 \cdots c_{n-1} \in \mathbb{F}^n \longleftrightarrow c(x) = c_0 + c_1 x + \cdots + c_{n-1} x^{n-1} \in \mathbb{F}[x].$$

Utilizando esta correspondencia, la condición de ciclicidad es equivalente la siguiente condición:

$$c(x) \in \mathcal{C} \implies xc(x) \bmod x^n - 1 \in \mathcal{C}.$$

Es natural pensar en usar el anillo cociente $\mathbb{F}[x]/(x^n - 1)$ para el estudio de los códigos cíclicos. En efecto, existe una correspondencia directa entre los códigos cíclicos y los ideales de $\mathbb{F}[x]/(x^n - 1)$.

Extensiones de Ore

Extensiones de Ore

Definición

Sea $\mathbb{F}$ un cuerpo cualquiera y σ un automorfismo de $\mathbb{F}$. Entonces, el anillo $R = \mathbb{F}[x;\sigma]$ de los polinomios usuales de $\mathbb{F}[x]$, cuyo producto verifica

$$xa = \sigma(a)x \quad \forall a \in \mathbb{F},$$

es una extensión de Ore o un anillo de polinomios sesgados.

Extensiones de Ore

Definición

Sea $\mathbb{F}$ un cuerpo cualquiera y σ un automorfismo de $\mathbb{F}$. Entonces, el anillo $R = \mathbb{F}[x;\sigma]$ de los polinomios usuales de $\mathbb{F}[x]$, cuyo producto verifica

$$xa = \sigma(a)x \quad \forall a \in \mathbb{F},$$

es una extensión de Ore o un anillo de polinomios sesgados.

Conceptos como el grado o el coeficiente lider se definen de la misma forma que en los anillos de polinomios usuales, y existen algoritmos de división tanto a izquierda como a derecha similares al de división euclídea usual.

Mínimo común múltiplo y máximo común divisor

De forma similar a como se hace para dominios euclideos, gracias a estos dos algoritmos podemos definir los conceptos de máximo común divisor y mínimo común múltiplo respectivos.

Mínimo común múltiplo y máximo común divisor

De forma similar a como se hace para dominios euclideos, gracias a estos dos algoritmos podemos definir los conceptos de máximo común divisor y mínimo común múltiplo respectivos.

Notaremos por $(f, g)_{l|r}$ al máximo común divisor a la izquierda o derecha respectivamente, y por $[f, g]_{l|r}$ al mínimo común múltiplo de igual forma.

Mínimo común múltiplo y máximo común divisor

De forma similar a como se hace para dominios euclideos, gracias a estos dos algoritmos podemos definir los conceptos de máximo común divisor y mínimo común múltiplo respectivos.

Notaremos por $(f, g)_{l|r}$ al máximo común divisor a la izquierda o derecha respectivamente, y por $[f, g]_{l|r}$ al mínimo común múltiplo de igual forma.

Existen también versiones del algoritmo extendido de Euclides a izquierda y derecha, que nos proporcionan los coeficientes de Bezout a izquierda o derecha.

Códigos cíclicos sesgados

Introducción

Utilizaremos la misma notación utilizada hasta ahora, con la única condición añadida de que σ tiene que ser de orden finito n.

Introducción

Utilizaremos la misma notación utilizada hasta ahora, con la única condición añadida de que σ tiene que ser de orden finito n.

Por ser el polinomio $x^n - 1$ central, podemos contruir el anillo cociente $\mathcal{R} = R/\langle x^n - 1 \rangle$, que será sobre el que trabajaremos de aquí en adelante.

Codigos cíclicos sesgados

Sea $\mathfrak{v} : \mathcal{R} \to \mathbb{F}^n$ el mapa de coordenadas asociado a la base $\{1, x, \ldots, x^{n-1}\}$ de $\mathcal{R}$.

Codigos cíclicos sesgados

Sea $\mathfrak{v} : \mathcal{R} \to \mathbb{F}^n$ el mapa de coordenadas asociado a la base $\{1, x, \ldots, x^{n-1}\}$ de $\mathcal{R}$.

Definición (Código cíclico sesgado)

Un código lineal $\mathcal{C}$ sobre $\mathbb{F}$ de longitud n es un código cíclico sesgado si y solo si es la imagen por $\mathfrak{v}$ de un ideal a la izquierda I de $\mathcal{R}$. Simbólicamente

$$\mathcal{C} = \mathfrak{v}(I).$$

Codigos cíclicos sesgados

Sea $\mathfrak{v} : \mathcal{R} \to \mathbb{F}^n$ el mapa de coordenadas asociado a la base $\{1, x, \ldots, x^{n-1}\}$ de $\mathcal{R}$.

Definición (Código cíclico sesgado)

Un código lineal $\mathcal{C}$ sobre $\mathbb{F}$ de longitud n es un código cíclico sesgado si y solo si es la imagen por $\mathfrak{v}$ de un ideal a la izquierda I de $\mathcal{R}$. Simbólicamente

$$\mathcal{C} = \mathfrak{v}(I).$$

Para cada código cíclico C existe un único polinomio mónico $g \in R$ que genere C (como ideal a la izquierda) y que divida a $x^n - 1$, al que llamaremos *polinomio generador*.

Codigos skew RS

Definición (Código skew RS)

Sea α un generador normal de $\mathbb{F}$ y $\beta = \alpha^{-1}\sigma(\alpha)$. Entonces, un código Reed-Solomon sesgado (o skew RS para abreviar) de distancia mínima diseñada δ es un código cíclico sesgado generado por

$$[x - \sigma^r(\beta), x - \sigma^{r+1}(\beta), \cdots, x - \sigma^{r+\delta-2}(\beta)]_l,$$

para algún $r \geq 0$.

Codigos skew RS

Definición (Código skew RS)

Sea α un generador normal de $\mathbb{F}$ y $\beta = \alpha^{-1}\sigma(\alpha)$. Entonces, un código Reed-Solomon sesgado (o skew RS para abreviar) de distancia mínima diseñada δ es un código cíclico sesgado generado por

$$[x - \sigma^r(\beta), x - \sigma^{r+1}(\beta), \cdots, x - \sigma^{r+\delta-2}(\beta)]_I,$$

para algún $r \geq 0$.

Nota: No se pierde generalidad al asumir $r = 0$.

Algoritmo de Sugiyama

Preámbulo

$\mathcal{C}$ denotará un código skew RS de distancia mínima δ generado por $[x-\beta, x-\sigma(\beta), \cdots, x-\sigma^{\delta-2}(\beta)]_l$, con capacidad para corregir $\tau = \lfloor(\delta-1)/2\rfloor$ errores.

Preámbulo

$\mathcal{C}$ denotará un código skew RS de distancia mínima δ generado por $[x-\beta, x-\sigma(\beta), \cdots, x-\sigma^{\delta-2}(\beta)]_l$, con capacidad para corregir $\tau = \lfloor(\delta-1)/2\rfloor$ errores.

Supongamos que se transmite una palabra $c \in \mathcal{C}$ por un canal ruidoso y que el polinomio $y = c + e$ es recibido, donde $e = e_1 x^{k_1} + \cdots + e_\nu x^{k_\nu}$ con $\nu \leq \tau$.

Polinomios localizador y evaluador de errores

Definimos el *polinomio localizador de errores* como

$$\lambda = \left[1 - \sigma^{k_1}(\beta)x, 1 - \sigma^{k_2}(\beta)x, \ldots, 1 - \sigma^{k_\nu}(\beta)x\right]_r.$$

Polinomios localizador y evaluador de errores

Definimos el *polinomio localizador de errores* como

$$\lambda = \left[1 - \sigma^{k_1}(\beta)x, 1 - \sigma^{k_2}(\beta)x, \ldots, 1 - \sigma^{k_\nu}(\beta)x\right]_r.$$

Este nos permite localizar las posiciones del error, pues $x - \sigma^{d-1}(\beta^{-1})$ divide a la izquierda a λ si y solo si d es una posición del error.

Polinomios localizar y evaluador de errores

Ahora, para cada $1 \leq j \leq \nu$ existe p_j tal que $\lambda = (1 - \sigma^{k_j}(\beta)x)p_j$. Definimos entonces el *polinomio evaluador de errores* como

$$\omega = \sum_{j=1}^{\nu} e_j \sigma^{k_j}(\alpha) p_j.$$

Polinomios localizar y evaluador de errores

Ahora, para cada $1 \leq j \leq \nu$ existe p_j tal que $\lambda = (1 - \sigma^{k_j}(\beta)x)p_j$. Definimos entonces el *polinomio evaluador de errores* como

$$\omega = \sum_{j=1}^{\nu} e_j \sigma^{k_j}(\alpha) p_j.$$

Conociendo el polinomio ω y los polinomios p_j podemos crear un sistema de ecuaciones lineal que nos permite calcular los valores del error en cada posición.

Síndromes y polinomio síndrome

Para cada $0 \leq i \leq n-1$, definimos el *i-ésimo síndrome* del polinomio recibido y como el resto de la división a la izquierda de y por $x - \sigma^i(\beta)$.

Síndromes y polinomio síndrome

Para cada $0 \leq i \leq n-1$, definimos el *i-ésimo síndrome* del polinomio recibido y como el resto de la división a la izquierda de y por $x - \sigma^i(\beta)$.

Se define el *polinomio síndrome* de y como:

$$S = \sum_{i=0}^{2\tau-1} \sigma^i(\alpha) S_i x^i.$$

Síndromes y polinomio síndrome

Para cada $0 \leq i \leq n-1$, definimos el *i-ésimo síndrome* del polinomio recibido y como el resto de la división a la izquierda de y por $x - \sigma^i(\beta)$.

Se define el *polinomio síndrome* de y como:

$$S = \sum_{i=0}^{2\tau-1} \sigma^i(\alpha) S_i x^i.$$

Además, si se da el caso $S = 0$, entonces $y \in C$.

Ecuación clave

Teorema

La ecuación clave no conmutativa

$$x^{2\tau}u + S\lambda = \omega$$

se cumple para un polinomio u de grado menor que ν, y además es un múltiplo a la derecha de la ecuación

$$x^{2\tau}u_l + Sv_l = r_l,$$

donde u_l, v_l y r_l son los coeficientes de Bezout dados por el algoritmo extendido de Euclides a la derecha con entrada $x^{2\tau}$ y S, e l es el índice determinado por las condiciones $\deg r_{l-1} \geq \tau$ *y* $\deg r_l < \tau$

Algoritmo

Entrada: el código $\mathcal{C}$, el mensaje recibido $y = (y_0, \ldots, y_{n-1}) \in \mathbb{F}^n$ con no más de τ errores

Salida: una palabra código $c' \in \mathcal{C}$ o un *error en la ecuación clave*

```
// Paso 1: calcular polinomio síndrome
para 0 ≤ i ≤ 2t − 1 hacer
    S_i ← Σ_{j=0}^{n−1} y_j N_j(σ^i(β))
fin
S ← Σ_{i=0}^{2τ−1} σ^i(α) S_i x^i
si S = 0 entonces
    devolver y
fin
```

Algoritmo

Entrada: el código $\mathcal{C}$, el mensaje recibido $y = (y_0, \ldots, y_{n-1}) \in \mathbb{F}^n$ con no más de τ errores

Salida: una palabra código $c' \in \mathcal{C}$ o un *error en la ecuación clave*

```
// Paso 1: calcular polinomio síndrome
```

para $0 \leq i \leq 2t - 1$ **hacer**
  $S_i \leftarrow \sum_{j=0}^{n-1} y_j N_j(\sigma^i(\beta))$
fin
$S \leftarrow \sum_{i=0}^{2\tau-1} \sigma^i(\alpha) S_i x^i$
si $S = 0$ **entonces**
  devolver y
fin

Algoritmo

```
   // Paso 2: aplicar REEA
 8 {u_i, v_i, r_i}_{i=0,...,l} ← REEA(x^{2τ}, S)
 9 l ← primera iteración en REEA tal que deg r_i < τ
10 pos ← ∅
   // Paso 3: localizar posiciones del error
11 para 0 ≤ i ≤ n − 1 hacer
12     si x − σ^{i−1}(β^{−1}) es un divisor a la izquierda de v_l entonces
13         pos = pos ∪ {i}
14     fin
15 fin
```

Algoritmo

```
   // Paso 2: aplicar REEA
 8 {u_i, v_i, r_i}_{i=0,...,l} ← REEA(x^{2τ}, S)
 9 l ← primera iteración en REEA tal que deg r_i < τ
10 pos ← ∅
   // Paso 3: localizar posiciones del error
11 para 0 ≤ i ≤ n − 1 hacer
12     si x − σ^{i−1}(β^{−1}) es un divisor a la izquierda de v_l entonces
13         pos = pos ∪ {i}
14     fin
15 fin
```

Algoritmo

```
   // Comprobamos si se han encontrado todas las posiciones del error
16 si deg v_l > Cardinal(pos) entonces
17 |   devolver error en la ecuación clave
18 fin
   // Paso 4: Construir y resolver el sistema
19 para j ∈ pos hacer
20 |   p_j ← rquo(v_l, 1 − σ^j(β)x)
21 fin
22 Resolver el sistema lineal r_l = Σ_{j∈pos} e_j σ^j(α) p_j
   // Paso 5: Eliminar el error del mensaje recibido
23 e ← Σ_{j∈pos} e_j x^j
24 devolver y − e
```

Algoritmo

// Comprobamos si se han encontrado todas las posiciones del error
si $\deg v_I > \text{Cardinal}(pos)$ **entonces**
  devolver *error en la ecuación clave*
fin
// Paso 4: Construir y resolver el sistema
para $j \in pos$ **hacer**
  $p_j \leftarrow \text{rquo}(v_I, 1 - \sigma^j(\beta)x)$
fin
Resolver el sistema lineal $r_I = \sum_{j \in pos} e_j \sigma^j(\alpha) p_j$
// Paso 5: Eliminar el error del mensaje recibido
$e \leftarrow \sum_{j \in pos} e_j x^j$
devolver $y - e$

Algoritmo

```
   // Comprobamos si se han encontrado todas las posiciones del error
16 si deg v_l > Cardinal(pos) entonces
17 |   devolver error en la ecuación clave
18 fin
   // Paso 4: Construir y resolver el sistema
19 para j ∈ pos hacer
20 |   p_j ← rquo(v_l, 1 − σ^j(β)x)
21 fin
22 Resolver el sistema lineal r_l = Σ_{j∈pos} e_j σ^j(α) p_j
   // Paso 5: Eliminar el error del mensaje recibido
23 e ← Σ_{j∈pos} e_j x^j
24 devolver y − e
```

Implementación en SageMath

Clases desarrolladas

Para la implementación se ha aprovechado la estructura de clases presente en SageMath para códigos. Se han desarrollado en este entorno:

- Clases para representar códigos cíclicos sesgados y códigos skew RS.
- Codificador en forma vectorial y polinomial de mensajes para códigos cíclicos sesgados.
- Decodificador para códigos skew RS utilizando el algoritmo de Sugiyama presentado.

Además, se ha implementado también las versiones a izquierda y derecha del algoritmo extendido de Euclides.

Ejemplo

Implementación en SageMath

Conclusiones

Conclusiones

- Objetivos:

Conclusiones

- Objetivos:
  - Estudio de los fundamentos de la teoría de códigos.

Conclusiones

- Objetivos:
  - Estudio de los fundamentos de la teoría de códigos.
  - Estudio de las extensiones de Ore.

Conclusiones

- Objetivos:
  - Estudio de los fundamentos de la teoría de códigos.
  - Estudio de las extensiones de Ore.
  - Estudio de los códigos cíclicos sesgados y del algoritmo de Sugiyama.

Conclusiones

- Objetivos:
  - Estudio de los fundamentos de la teoría de códigos.
  - Estudio de las extensiones de Ore.
  - Estudio de los códigos cíclicos sesgados y del algoritmo de Sugiyama.
  - Implementación en SageMath.

Conclusiones

- Objetivos:
  - Estudio de los fundamentos de la teoría de códigos.
  - Estudio de las extensiones de Ore.
  - Estudio de los códigos cíclicos sesgados y del algoritmo de Sugiyama.
  - Implementación en SageMath.
- Posible trabajo futuro: completar las clases desarrolladas y contribuir al proyecto SageMath.