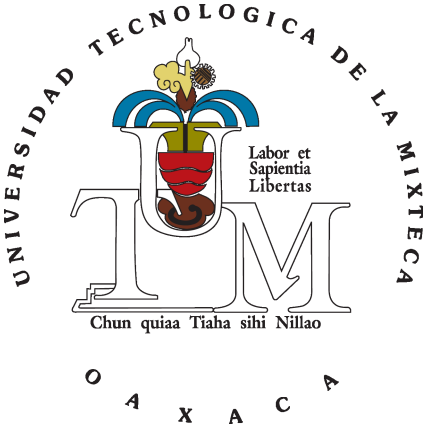


UNIVERSIDAD TECNOLÓGICA DE LA MIXTECA

Códigos cíclicos basados en campos finitos para desarrollar algoritmos de ADN

TESIS

PARA OBTENER EL TÍTULO DE:

Licenciada en Matemáticas Aplicadas

PRESENTA:

Nayeli Adriana González Martínez

DIRECTOR DE TESIS:

Dr. Adolfo Maceda Méndez

Huajuapan de León, Oaxaca, México Enero del 2019

Dedicado a

A mi adorada hija

Quien es mi motor y mi estímulo para seguir adelante, quien con sus risas y sus travesuras alegra mi vida, y me impulsa a seguir trabajando y superándome.

A mi amado esposo

Quien con su amor, y apoyo incondicional, siempre me motivó para concluir este trabajo, creyó en mí, en mis capacidades y habilidades incluso cuando yo dudaba de ellas.

A mis queridos padres

Pues su valioso apoyo y su confianza en mí me ayudaron e impulsaron a salir adelante durante mi carrera, y a quienes debo lo que hoy soy .

A mi hermano

Quien además es mi amigo y siempre me brindó su apoyo y su cariño.

Agradecimientos

A Dios, por darme la vida, por guiarme siempre, por poner en mi camino a personas increíbles y por permitirme la dicha de ser mamá de una nena maravillosa.

A mi director de tesis, el doctor Adolfo Maceda Méndez, por su apoyo y paciencia durante el desarrollo de esta tesis, por darme la oportunidad de trabajar con usted y por el valioso tiempo que dedicó a este trabajo, por sus consejos y sugerencias, por su guía y su comprensión cuando decidí posponer mi trabajo.

A mis padres, Lorena y Natalio, desde niña me enseñaron a esforzarme para alcanzar mis metas, estuvieron conmigo a cada momento, impulsándome y apoyándome. Ustedes siempre creyeron en mí y confiaron en mis capacidades, apoyaron con amor cada decisión que tomé por difícil que fuera para ustedes aceptarla. Cuando me alejé de casa para seguir mis sueños y estudiar la carrera que yo quería, confiaron en mí y me convencieron de que yo podía lograrlo, sin ustedes no habría sido posible lograr este sueño.

A mi maravilloso esposo, Alan, desde que te conocí me has dado lo mejor de tí, me haz hecho crecer, me haz motivado a superarme y has visto en mí cualidades que ni yo misma veía. Haz estado conmigo en las buenas y en las malas, haz sido mi apoyo incondicional. Gracias por creer en mí, por amarme tanto y por no dejarme renunciar nunca.

A mi pequeña Amy, tú eres un ángel en mi vida, eres mi razón para superarme, eres mi pequeño milagro, gracias amor por tus tiernas palabras:"No te preocupes mami, yo estoy contigo", siempre serás mi luz.

A mi hermano Omar, por tu apoyo y tu ayuda, por estar conmigo cuando te necesito y escucharme siempre.

Finalmente quiero agradecer a todos aquellos que me apoyaron durante la realización de este trabajo, en especial a mis revisores, la doctora Luz del Carmen Álvarez Marín, el maestro Mario Lomelí Haro, y el maestro Vulfrano Tochihuitl Bueno, quienes se tomaron el tiempo para la revisión de este trabajo, por sus comentarios y sus sugerencias.

Introducción

La idea de usar ADN para almacenar y procesar información despegó en 1994 cuando un científico de California llamado Leonard Adleman utilizó por primera vez el ADN en un tubo de ensayo para resolver un problema matemático simple. La computación del ADN es una forma de computación que usa ADN y moléculas biológicas, en lugar de las tecnologías informáticas tradicionales basadas en silicio, una de las razones por las que empezó a ser interesante estudiar la computación del ADN es que un solo gramo de ADN con un volumen de 1 cm^3 puede contener tanta información que un billón de discos compactos, aproximadamente 750 *terabytes*, por ello la capacidad computacional de los sistemas vivientes ha intrigado por años a los investigadores, de ahí la motivación para implementar e imitar los procesos biológicos en sistemas computacionales, [15].

Otra característica importante de los procesos entre el ADN, que llama la atención de los investigadores, es la capacidad de realizar muchos procesos en paralelo, esto es posible gracias a la capacidad de replicación del ADN, y constituye una importante ventaja al realizar cálculos muy grandes, puesto que disminuye el tiempo de respuesta, al hacer simultáneamente varios procesos. De aquí el principal interés de utilizar la computación del ADN para resolver problemas matemáticos de gran complejidad.

Richard Feynman introdujo por primera vez el cálculo molecular a principios de 1960. Sin embargo, el primero en desarrollar en la práctica este tipo de computación fue Leonard Adleman de la Universidad del Sur de California. En 1994, Adleman demostró un uso del ADN como forma de cálculo para resolver el problema del camino hamiltoniano de siete puntos, este problema consiste en hallar un camino hamiltoniano dada una gráfica, es decir, hallar una ruta que pasa por todos los vértices de la gráfica exactamente una vez; este problema debido a su complejidad pertenece a los problemas denominados NP-Completos, los cuales son muy difíciles de resolver pues no se pueden resolver en tiempo polinomial.

Después del trabajo de Adleman, diversos investigadores empezaron a encontrar nuevas aplicaciones para la computación del ADN, por ejemplo, en 1995, Lipton propuso experimentos de ADN para resolver el problema de satisfacibilidad: Dado un conjunto de cláusulas, ¿Existe un conjunto de valores booleanos para una determinada expresión que la haga verdadera? Lipton consiguió

establecer un algoritmo de ADN para resolver cualquier instancia del problema de satisfacibilidad de tamaño prefijado. Por lo tanto, fue el primero en establecer un esquema algorítmico molecular.

En otras áreas de investigación también han habido importantes aplicaciones de la computación del ADN, no sólo en matemáticas, puesto que muchos investigadores han notado las ventajas de llevar sus cálculos a nivel molecular.

A pesar de las grandes ventajas que ofrece la computación del ADN, su potencial es limitado precisamente por la composición química del ADN, debido a que el ADN es una doble hélice formada por una hebra simple de ADN y su complemento Watson-Crick, unidas mediante un proceso llamado hibridación específica, al formarse esta doble hélice pueden haber errores y dar como resultado cadenas de ADN que no funcionan, es decir, se obtienen falsos positivos o falsos negativos. Varios autores han propuesto diferentes técnicas para construir un conjunto de palabras código de ADN que es poco probable que formen enlaces indeseables entre sí por hibridación.

El objetivo de este trabajo de tesis es usar la teoría de los códigos cíclicos para construir códigos que son adecuados para la computación del ADN. En particular, construimos códigos cíclicos lineales y aditivos sobre $GF(4)$ que son adecuados para esta aplicación. Además la principal ventaja de los códigos aditivos sobre los códigos lineales es que, para la misma longitud del código, hay más códigos aditivos que códigos lineales. Lo que permite encontrar códigos aditivos que tengan más palabras código que un código lineal de la misma longitud pero con la misma distancia de Hamming; lo que a su vez permite, incrementar el número de secuencias válidas de ADN, y por consecuencia permite tener más almacenamiento.

Este trabajo está estructurado como sigue: En el capítulo 1, exponemos la teoría básica de códigos que utilizaremos para entender cómo son los códigos que mejorarán los algoritmos de ADN. En el capítulo 2, hacemos un recordatorio de la teoría de álgebra que permite generar los códigos cíclicos aditivos. En el capítulo 3, desarrollamos la teoría necesaria para hallar los códigos cíclicos aditivos que mejoran los algoritmos de ADN, basándonos en el artículo de Abualrub [2]. Por último en el capítulo 4 de este trabajo explicamos brevemente algunas aplicaciones de la computación del ADN en teoría de gráficas, donde se ha utilizado la computación del ADN para resolver problemas importantes de matemáticas.

Índice general

Capítulo 1

Elementos básicos de teoría de códigos

El proceso de transmisión e intercambio de conocimientos, datos, e información general es indispensable para entender no sólo la evolución y desarrollo del ser humano como especie sino también para entender su realidad.

Cuando deseamos transmitir un mensaje debemos saber que en el proceso, el mensaje puede ser alterado, debido a las interferencias que hay en el canal en que se transmite. Por ejemplo, en una llamada telefónica puede haber ruido que no permite que el receptor reciba el mensaje que originalmente envía el emisor. Para lograr que el mensaje que se ha enviado sea el correcto, es importante conocer el proceso de la transmisión de la información. Al saber esto, quizá podamos detectar cuándo un mensaje ha sido alterado y en algunas ocasiones corregirlo.

En el proceso de la transmisión de información de un sistema de comunicación digital se llevan a cabo las siguientes tareas:

1. Codificación fuente
2. Canal de codificación
3. Modulación

El receptor del correspondiente mensaje realiza:

1. Demodulación
2. Canal de decodificación
3. Decodificación fuente

El modulador genera la señal que es usada para transmitir una secuencia de símbolos a través del canal; debido al ruido en el canal, la señal transmitida es perturbada. El ruido recibido en la señal es demodulado por el receptor para obtener la secuencia de símbolos recibidos. Como la secuencia de símbolos recibidos usualmente difiere de los símbolos transmitidos, el receptor puede utilizar el canal de codificación para detectar e incluso corregir algunos errores. Para terminar, el canal codificador introduce redundancias a la información original. Las redundancias son explotadas por el canal decodificador para la detección o corrección de errores por la estimación de los símbolos transmitidos.

Un ejemplo de este proceso sería el siguiente; supongamos que se quiere trasmitir un mensaje que consiste en la palabra SI, codificamos $SI = 1$ y $NO = 0$, pero durante el envío de la información el mensaje es alterado y en lugar de recibir 1 recibimos 0, entonces obtendríamos un mensaje erróneo al decodificar. Para evitar esto podríamos agregar redundancias o repeticiones a la codificación y codificar $SI = 11$ y $NO = 00$, así si recibimos el mensaje 01 podemos asegurar que el mensaje recibido es equivocado y solicitar que el emisor lo envíe de nuevo. Notemos que esta nueva codificación nos permite identificar errores más no corregirlos pues de igual manera el mensaje original podría ser 00 o 11.

1.1. Fundamentos de códigos de bloque

Una de las definiciones fundamentales para entender los códigos es la siguiente.

Definición 1.1.1. *Un alfabeto es un conjunto finito $\mathcal{A} = \{a_1, \ldots, a_q\}$, cuyos elementos son llamados símbolos y q (número de elementos de A) es la raíz de $\mathcal{A}$.*

A partir de los elementos de un alfabeto, tenemos la siguiente definición.

Definición 1.1.2. *Una palabra de longitud n sobre $\mathcal{A}$ es una sucesión de n elementos de $\mathcal{A}$; en general escribiremos las palabras de la siguiente forma,*

$$a = a_{i_1} a_{i_2} \ldots a_{i_n}, \quad a_{i_k} \in \mathcal{A}$$

Denotaremos por $\mathcal{A}^n$ a todas las palabras de longitud n y por $\mathcal{A}^*$ al conjunto que contiene a todas las palabras, es decir, $\mathcal{A}^* = \bigcup_{n \in \mathbb{N}} \mathcal{A}^n$.

Definición 1.1.3. *Si $\mathcal{A} = \{a_1, \ldots, a_q\}$ es un alfabeto, un código q-ario sobre $\mathcal{A}$ es un subconjunto C de $\mathcal{A}^*$. Los elementos de C son llamadas palabras códigos o codewords. El número $M = |C|$, el cardinal de C, es llamado tamaño del código. Podemos distinguir dos tipos de código según la longitud de sus palabras,*

- *Códigos de bloque: La longitud de todas las palabras del código es la misma y se dice que tiene parámetros (n, M), donde M es el tamaño del código y n es la longitud de las palabras código.*

- *Códigos de longitud variable: Está conformado por palabras de diferente longitud.*

Como los únicos códigos que trataremos en este trabajo son los códigos de longitud fija, siempre que hablemos de códigos serán considerados códigos de bloque.

Consideremos una secuencia de información $u_0u_1u_2u_3u_4u_5u_6u_7\ldots$ de símbolos de información u_i tomados de una alfabeto $\mathcal{A}$. Esta secuencia de información es agrupada en bloques de longitud k de acuerdo a:

$$\underbrace{u_0u_1\cdots u_{k-1}}\underbrace{u_ku_{k+1}\cdots u_{2k-1}}\underbrace{u_{2k}u_{2k+1}\cdots u_{3k-1}}\cdots .$$

Cada uno de estos bloques de información forman una palabra a la cual denominaremos como *palabra de información de longitud k*. Ahora veamos como se forma un código de bloque teniendo una secuencia de palabras de información.

Definición 1.1.4. *Un $q-(n,k)$ código de bloque es un código cuyas palabras de información son:*

$$\begin{aligned}
&u_0u_1\cdots u_{k-1},\\
&u_ku_{k+1}\cdots u_{2k-1},\\
&u_{2k}u_{2k+1}\cdots u_{3k-1},\\
&\vdots
\end{aligned}$$

de longitud k con $u_i \in \mathcal{A},\ |\mathcal{A}| = q$, cada una de estas palabras es codificada en cada una de las siguientes palabras código:

$$\begin{aligned}
&b_0b_1\cdots b_{n-1},\\
&b_nb_{n+1}\cdots b_{2n-1},\\
&b_{2n}b_{2n+1}\cdots b_{3n-1},\\
&\vdots
\end{aligned}$$

de longitud n con $b_i \in \mathcal{A},\ |\mathcal{A}| = q$.

Las palabras código son transmitidas a través del canal y las palabras recibidas son apropiadamente decodificadas. Como cada símbolo de las palabras de información puede tomar uno de los q posibles valores, el *número de palabras de información es q^k*.

Además notemos que cada una de las palabras de información de longitud k son transformadas en palabras código de longitud n, donde $n \geq k$ puesto que al pasar una palabra de información a una palabra código se agregan redundancias para evitar así errores en la transmisión de la información,

es decir, a las palabras de información se les introducen deliberadamente símbolos del alfabeto que no son parte de la información y que sirven para detectar posibles errores.

Los códigos son caracterizados principalmente por dos parámetros llamados *tasa del código* y *mínima distancia de Hamming* los cuales son utilizados para evaluar la eficiencia de codificación y la capacidad de detectar y corregir errores.

1.1.1. Tasa del código

Bajo el supuesto de que cada símbolo de información u_i, con $i \in \{0, 1, \ldots, k-1\}$ de un (n, M) código de bloque puede asumir q valores, el *número de palabras código* está dado por

$$M = q^k.$$

Como cada palabra código de longitud n es más larga que las palabras de información de longitud k, la tasa a la cual la información es transmitida a través del canal es reducida por la llamada *tasa del código*

$$R = \frac{\log_q(M)}{n} = \frac{k}{n}.$$

1.1.2. Distancia de Hamming

Para explicar cómo funciona la correción y detección de errores en los códigos pongamos el siguiente ejemplo.

Supongamos que la persona A debe reunirse con la persona B y ambos tienen el mismo mapa, pero sólo B sabe el camino a seguir. Suponemos que B tiene que transmitir el mensaje $NNOONOOO$, el alfabeto utilizado es N, S, E, O que son las iniciales de los puntos cardinales. Esto se podría codificar en palabras código binarias; el código de menor tamaño sería $C_1 = \{00, 01, 10, 11\}$ con $N = 00, S = 01, E = 10, O = 11$.

Note que si usamos esta codificación y recibimos la palabra 01 que corresponde a S en lugar de 00 que es N, como $01 \in C_1$, es decir, es una palabra válida, no es posible saber si la palabra es correcta o no.

En cambio, si utilizamos el código $C_2 = \{000, 011, 101, 110\}$ con la codificación $N = 000, S = 011, E = 101, O = 110$, si recibimos por ejemplo 111, esta palabra no pertenece a C_2, y por tanto no es una palabra válida, así sabemos que el mensaje recibido es incorrecto, pero no hay forma de saber cuál es la palabra adecuada, puesto que podría ser $011, 101$ o bien 110. Esto quiere decir que el código C_2 permite detectar un error pero no permite corregirlo.

Para poder corregir un error debemos agregar redundancias al código, es decir, podemos utilizar el código $C_3 = \{00000, 01101, 10110, 11011\}$ con la codificación $N = 00000, S = 01101, E = 10110, O = 11011$. Si recibimos por ejemplo 11111 nos damos cuenta que no es palabra válida

y además notamos que la palabra más cercana es $O = 11011$, así que podemos corregir el error detectado.

Todo esto de la detección y corrección de errores es posible gracias a un parámetro que nos da una idea de que tan cerca está una palabra de otra, como se explica a continuación.

La distancia entre dos palabras código $\mathbf{b} = b_0 b_1 \cdots b_{n-1}$ y $\mathbf{b}' = b'_0 b'_1 \cdots b'_{n-1}$ está dada por la llamada *distancia de Hamming* denotada por $H(\mathbf{b}, \mathbf{b}')$,

$$H(\mathbf{b}, \mathbf{b}') = |\{0 \leq i < n : b_i \neq b'_i\}|.$$

El resultado siguiente dice que la distancia de Hamming es una métrica en el conjunto $\mathcal{A}^n$. (El lector interesado en ver la demostración puede consultar [14])

Teorema 1.1.1. *La función $H : \mathcal{A}^n \times \mathcal{A}^n \to [0, n] \subset \mathbb{N}$, donde, $H(x, y) = |\{1 \leq i \leq n : x_i \neq y_i\}|$, es una métrica en $\mathcal{A}^n$, es decir, satisface,*

1. *$H(x, y) \geq 0$ y $H(x, y) = 0$ si y sólo si $x = y$.*
2. *$H(x, y) = H(y, x)$.*
3. *$H(x, z) \leq H(x, y) + H(y, z)$.*

Ahora daremos la definición de distancia de un código.

Definición 1.1.5. *Dado un código C, se define la distancia de C, y se denota por $d(C)$, como la menor distancia no nula entre sus palabras código, es decir,*

$$d(C) = mín\{H(x, y) : x, y \in C,\ x \neq y\}.$$

Esta distancia es utilizada para hallar el número de errores detectables en el código, para ello requerimos la siguiente definición y enunciaremos un teorema que nos ayuda a calcular dicho número.

Definición 1.1.6. *Sea e un entero positivo. El código C se dice que corrige hasta e errores (C es un código $e-$corrector) si se cumple que para cualquier palabra w existe a lo más una palabra $c \in C$ tal que $H(w, c) = e$.*

Definición 1.1.7. *Sea e un entero positivo. El código C se dice que detecta hasta e errores (C es un código $e-$detector) si se cumple que para cualquier palabra w recibida en un mensaje codificado, existe al menos una palabra $c \in C$ tal que $H(w, c) \leq e$.*

Teorema 1.1.2. *Sea e un entero positivo y C un código.*

(i) C detecta hasta e errores en cualquier palabra código si $d(C) \geq e + 1$.

(ii) C corrige hasta e errores si y sólo si $d(C) \geq 2e + 1$.

Demostración. Sea C un código y e un entero positivo,

(i) Supongamos que $d(C) \geq e+1$ y que se transmite la palabra código c y se producen e errores o menos. La palabra recibida es c' y entonces

$$H(c, c') \leq e.$$

Notemos que c' no puede estar en el código y por lo tanto la detectamos como errónea.
Si c' estuviera en C tendría que ocurrir que $H(c, c') \geq d(C)$ y $H(c, c') \leq e$, entonces $d(C) \leq e \leq d(C) - 1$, lo cual es una contradicción.

Por tanto el código C puede detectar hasta e errores.

(ii) Supongamos que $d(C) \geq 2e + 1$. Si existe una palabra w de forma que existen $c_1, c_2 \in C$ distintas, con $H(c_1, w) \leq e$ y $H(c_2, w) \leq e$, entonces $H(c_1, c_2) \leq 2e$, lo cual es una contradicción, así C corrige hasta e errores.

Recíprocamente, supongamos que C corrige hasta e errores y $d(C) < 2e+1$, es decir, $d(C) \leq 2e$, consideremos la parte entera f de $d(C)/2$. Entonces $f \leq d(C)/2 \leq e$, además $d(C)-f \leq e$, ya que si $d(C)$ es par $f = d(C)/2$ y si $d(C)$ es impar, $d(C) - f = 2f + 1 - f = f + 1 = (d(C) + 1)/2 \leq e$, pues $d(C) + 1 \leq 2e$.

Sean $c_1, c_2 \in C$ tales que $H(c_1, c_2) = d(C) = d$, y sea w la palabra obtenida cambiando f coordenadas de c_1 de las d en que difieren c_1 y c_2 hasta que coincidan con sus correspondientes en c_2, así $H(c_1, w) = f \leq e$, entonces $H(c_2, w) = d - f \leq e$ y C no corregiría e errores

□

Resumiendo, el número de errores detectables en un código es calculado mediante,

$$e_{det} = d(C) - 1.$$

Y el número de errores que se pueden corregir en el código está dado por

$$e_{cor} = \lfloor \frac{d(C) - 1}{2} \rfloor,$$

donde $\lfloor z \rfloor$ denota el mayor número entero que es menor que z.

Ejemplo. *Consideremos el código* $C = \{001, 010, 100\}$ *cuya distancia es* $d = 2$. *Aplicando el resultado anterior tenemos que este código detecta* 1 *error,*

- *Un error aislado en* C *se puede detectar.*
  - *Si ocurre un error en* 001*, se recibiría una de las siguientes palabras,* $101, 011, 000$*, sin embargo ninguna de ellas pertenece al código, por tanto se detecta el error.*

- *Dos errores aislados no pueden detectarse.*
  - *Si ocurren dos errores en* 001*, se recibiría una de las siguientes palabras,* $111, 100, 010$*, pero dos de ellas pertenecen a* C*, por lo tanto puede ocurrir que no notemos el error.*

Ahora veamos el siguiente ejemplo para entender qué pasa con los códigos correctores.

Ejemplo. *Consideremos el código*

$$C = \{0000000000, 0000011111, 1111100000, 1111111111\},$$

cuya distancia es $d = 5$*. Aplicando el resultado anterior, se tiene lo siguiente,*

- *El código* C *detecta* $d - 1 = 4$ *errores.*
  - *Si ocurren* 4 *errores en* 0000011111*, se podrían recibir palabras como,* 1111011111, 0111111111, 0001100111, 0000000001, 0000010000 *y si revisamos ninguna palabra de este tipo está en* C*.*
- *El código* C *corrige* $\lfloor \frac{d-1}{2} \rfloor = 2$ *errores.*
  - *Si recibimos la palabra* 0000011100 *con errores en los dos últimos dígitos, notamos de inmediato que se recibió una palabra errónea, y notamos que la palabra que está más cercana y que está además en el código es* 0000011111*, por lo que podemos corregirla.*
  - *Si recibimos* 0000011000 *con errores en los tres últimos dígitos, podemos notar también que se ha recibido una palabra equivocada, sin embargo, hay una palabra en el código que está más cerca de ella que la palabra correcta. Así podríamos pensar que la palabra correcta es* 0000000000*, cuando en realidad es* 0000011111*. Por tanto aquí ya no es posible corregir el error, sólo podemos detectarlo.*

Capítulo 2

Códigos basados en campos finitos

A lo largo de este capítulo, se construirán códigos basados en campos finitos, y se verá la importancia del álgebra lineal para el estudio de dichos códigos. También se utilizarán anillos de polinomios para estudiar propiedades de los códigos.

2.1. Anillos e ideales

En adelante utilizaremos frecuentemente la palabra *anillo*, cuya definición es la siguiente.

Definición 2.1.1. *Un anillo $\langle R, +, \cdot \rangle$ es un conjunto R junto con dos operaciones binarias $+$ y $\cdot$, que llamamos suma y multiplicación, definidas en R tales que se satisfacen los siguientes axiomas:*

- *$\langle R, + \rangle$ es un grupo abeliano.*
- *Para todo $a,\ b \in R$, $ab \in R$.*
- *La multiplicación es asociativa.*
- *Para todas las $a, b, c \in R$, se cumple la ley distributiva izquierda $a(b + c) = (ab) + (ac)$ y la ley distributiva derecha $(a + b)c = (ac) + (bc)$.*

Un anillo donde la multiplicación es conmutativa es llamado *anillo conmutativo.*

Definición 2.1.2. *Un anillo R con identidad multiplicativa 1 tal que $1x = x1 = x$ para todas las $x \in R$ se llama anillo unitario.*

Si R es un anillo unitario, entonces el elemento unitario 1 es único.

En un anillo unitario, el conjunto R^* de todos los elementos distintos de cero será un grupo multiplicativo si es cerrado bajo la multiplicación del anillo y si existen los inversos.

Definición 2.1.3. *Un inverso multiplicativo de un elemento a en un anillo R con unitario 1 es un elemento a^{-1} en R tal que $aa^{-1} = a^{-1}a = 1$. Si este elemento existe es único.*

Otra definición que se usará más adelante es la de *ideal*:

Definición 2.1.4. *Un ideal de R es un subgrupo aditivo $\langle N, + \rangle$ de un anillo R que satisface $rN \subseteq N$ y $Nr \subseteq N$ para todas las $r \in R$, donde $rN = \{rn : n \in N\}$ y $Nr = \{nr : n \in N\}$.*

Definición 2.1.5. *Si R es un anillo conmutativo unitario y $a \in R$, el ideal $\{ra | r \in R\}$ de todos los múltiplos de a es el ideal principal generado por a y se denota por $\langle a \rangle$. Un ideal N de R es un ideal principal si $N = \langle a \rangle$ para algún $a \in R$.*

Note que si $A = \langle a \rangle$ y $B = \langle b \rangle$ son ideales principales de un anillo R, y además $A \subseteq B$, entonces $a = br$, para algún $r \in R$, es decir, a es un múltiplo de b.

Enseguida daremos la definición de clase de un ideal.

Definición 2.1.6. *Sea N un ideal de un anillo R y sea $a \in R$. La clase $a + N$ de N es el conjunto $\{a + n : n \in N\}$.*

Dado que R es un anillo, con la operación de suma R es un grupo abeliano, por lo tanto, la suma de clases de un ideal I de R se define como:

$$(a + I) + (b + I) := (a + b) + I.$$

Adicionalmente se define el producto de clases como:

$$(a + I)(b + I) := (ab) + I.$$

Se establece que el producto y la suma están bien definidos puesto que no dependen de la elección de los representantes de cada clase.

Definición 2.1.7. *Si N es un ideal en un anillo R, entonces, el anillo de las clases $r + N$ bajo las operaciones inducidas es el anillo cociente, o el anillo factor, o el anillo de las clases residuales de R módulo N, y se denota por R/N. Las clases $r + N$ son las clases residuales módulo N.*

Enseguida daremos algunas definiciones que nos permitirán definir qué es un campo, concepto que utilizaremos frecuentemente en las siguientes secciones.

Definición 2.1.8. *Sea R un anillo con unitario. Un elemento u en R es una unidad de R si tiene inverso multiplicativo en R. Si todo elemento distinto de cero en R es una unidad entonces R es un semicampo o anillo con división.*

Definición 2.1.9. *Un campo es un anillo conmutativo con división.*

Definición 2.1.10. *Un ideal $M \neq R$ de un anillo R es maximal si no existe un ideal propio N de R que contenga propiamente a M.*

Teorema 2.1.1. *Sea R un anillo conmutativo con unitario. Entonces M es un ideal maximal si y solo si R/M es un campo.*

Entre anillos podemos tener funciones especiales que preserven ciertas características, estas funciones se llaman homomorfismos de anillos.

Definición 2.1.11. *Una función ϕ de un anillo R en un anillo R' es un homomorfismo si para cualesquiera a y b en R ocurre,*

$$\phi(a+b) = \phi(a) + \phi(b), \quad \phi(ab) = \phi(a)\phi(b)$$

Definición 2.1.12. *El núcleo o kernel de una función $\phi : R \to R'$ es el conjunto $\{r \in R : \phi(r) = 0\}$. Denotaremos a este conjunto como $ker(\phi)$.*

Nota. *El kernel de un homomorfismo $\phi : R \to R'$ es un ideal de R.*

Cuando un homomorfismo es $1-1$ o sobre se le denomina con un nombre especial como se enuncia en la siguiente definición.

Definición 2.1.13. *Sea ϕ un homomorfismo de R en R', si ϕ es $1-1$ es llamado monomorfismo; si ϕ es sobre, se le llama epimorfismo. Si ϕ es un monomorfismo y un epimorfismo al mismo tiempo, a ϕ se le llama isomorfismo; cuando esto ocurre decimos que R y R' son isomorfos y se denota por $R \simeq R'$.*

Uno de los teoremas más importantes que permite conservar propiedades de un anillo bajo un homomorfismo es el siguiente (vea la demostración en [7]).

Teorema 2.1.2. *Sea ϕ un homomorfismo de un anillo R en un anillo R'. Si 0 es la identidad aditiva en R entonces $\phi(0) = 0'$ es la identidad aditiva en R' y si $a \in R$, entonces $\phi(-a) = -\phi(a)$. Si S es un subanilllo de R, entonces $\phi(S)$ es un subanillo de R', y si S es un ideal de R implica que $\phi(S)$ es ideal de $\phi(R)$. Ahora en el otro sentido, si S' es un subanillo de R', entonces $\phi^{-1}(S')$ es un subanillo de R y S' ideal de $\phi(R)$ implica que $\phi^{-1}(S')$ es un ideal de R. Por último, si R tiene unitario 1 y $\phi(1) \neq 0'$ entonces, $\phi(1) = 1'$ es unitario para $\phi(R)$.*

Antes de concluir esta sección veamos un teorema muy importante en la teoría de anillos, llamado *Teorema de correspondencia*, pues muestra una relacion entre los ideales en un anillo y los ideales en el anillo cociente.

Teorema 2.1.3. *Sea $\varphi : A \to A/I$ la proyección canónica. Existe una correspondencia biunívoca entre ideales de A/I e ideales de A que contienen a I. Esta correspondencia está definida de la siguiente forma, $\phi(J) = \varphi(J)$, para todo ideal J de R que contiene a I.*

Demostración. Sea J un ideal del cociente. Entonces $\varphi^{-1}(J) = \phi^{-1}(J)$ es un ideal de A que contiene a I.

Del mismo modo $\varphi(\varphi^{-1}(J)) = \phi(\phi^{-1}(J)) = J$ es un ideal por ser la proyección canónica un epimorfismo. Esto prueba que la correspondencia es biunívoca.

□

2.2. Anillo de polinomios

Para iniciar esta sección definiremos primero qué es un polinomio, pues será importante saberlo para describir el anillo que utilizaremos. En adelante se supondrá que $0 \in \mathbb{N}$.

Definición 2.2.1. *Dado un anillo conmutativo con unitario R, un polinomio con coeficientes en R es una función $f : \mathbb{N} \to R$ para la cual existe $N \in \mathbb{N}$ con $f(n) = 0$ para todo $n > N$. Si $f(n) \neq 0$ para algún $n \in \mathbb{N}$, al máximo valor que satisface esto se le llama el grado de f, el cual se denota por* $\text{grad}(f)$.

Si f y g son polinomios con coeficientes en R se definen su suma $f + g$ y su producto fg como sigue:

$$(f + g)(n) = f(n) + g(n), (fg)(n) = \sum_{k=0}^{n} f(k)g(n - k)$$

Se verifica que $f + g$ y fg son polinomios y que si 0 representa la función constante 0, mientras que 1 representa la función definida como $1(0) = 1$, $1(n) = 0$ para toda $n > 0$, entonces el conjunto de polinomios con coeficientes en R es un anillo conmutativo con identidad.

Denotando por x a la función dada por $x(1) = 1$ y $x(n) = 0$ para cada $n \neq 1$, se verifica que, con la operación de multiplicación definida previamente, la función x^k satisface $x^k(k) = 1$ y $x^k(n) = 0$ si $n \neq k$. En consecuencia, todo polinomio distinto de cero se puede escribir en la forma

$$f = a_0 + a_1x + a_2x^2 + \cdots + a_nx^n.$$

En adelante denotaremos como $f(x)$ al polinomio f y $R[x]$ al conjunto de todos los polinomios con coeficientes en el anillo R, este conjunto es un anillo bajo la suma y multiplicación polinomial. Si R es conmutativo, entonces lo es también $R[x]$ y si R tiene unitario 1, entonces 1 también es unitario en $R[x]$.

Además cuando el anillo de coeficientes es también un campo se tiene una propiedad importante, como lo dice el siguiente teorema (vea [7]).

Teorema 2.2.1. *Si $\mathbb{K}$ es un campo, todo ideal en $\mathbb{K}[x]$ es principal.*

En adelante siempre que escribamos $\mathbb{K}$, estaremos hablando de un campo.

El siguiente teorema, cuya demostración puede encontrarse en [8], nos muestra como es el anillo de polinomios con coeficientes en un anillo cociente.

Teorema 2.2.2. *Si R es un anillo e I un ideal de R, entonces $I[x]$, el anillo de polinomios en x sobre I, es un ideal de $R[x]$. Además $R[x]/I[x] \simeq (R/I)[x]$ el anillo de polinomios en x sobre R/I.*

Teorema 2.2.3. *Si $\mathbb{K}$ es un campo e I el ideal principal de $\mathbb{K}[x]$ generado por el polinomio $p(x) = a_0 + a_1x + \cdots + a_nx^n$ con $a_n \neq 0$, entonces para cada $[r(x)] \in \mathbb{K}[x]/I$, existe un único polinomio $q(x) = b_0 + b_1x + \cdots + b_{n-1}x^{n-1}$ tal que $[r(x)] = [q(x)]$.*

Demostración. Sea $r(x) \in \mathbb{K}[x]$, aplicando el algoritmo de la división a $r(x)$ y $p(x)$, existe un único polinomio $g(x) \in \mathbb{K}[x]$, tal que,

$$r(x) = p(x)g(x) + q(x),$$

con $q(x) = 0$, o bien $grad(q(x)) < grad(p(x))$, luego,

$$[r(x)] = [p(x)g(x)] + [q(x)],$$

como $p(x)g(x) \in I$, entonces $[p(x)g(x)] = p(x)g(x) + I = I = [0]$, es decir, existe un polinomio $q(x)$ de grado menor que n, tal que,

$$[r(x)] = [q(x)].$$

Ahora veamos que $q(x)$ es único. Sea $q_1(x)$ con grado $n-1$, tal que $[r(x)] = [q_1(x)]$, entonces $[q_1(x)] = [q(x)]$, esto es $q_1(x) - q(x) \in I$, luego, $q_1(x) - q(x) = p(x)f(x)$, como el grado de $q_1(x) - q(x)$ es menor o igual que $n-1$ y el grado de $p(x)$ es n, debería ocurrir que $n + grad(f(x)) = n - 1$, pero esto no es posible. Por lo tanto el polinomio $q(x)$ es único. □

Definición 2.2.2. *Sea $\mathbb{K}$ un campo, un polinomio no constante $f(x) \in \mathbb{K}[x]$ es irreducible sobre $\mathbb{K}$ o es un polinomio irreducible en $\mathbb{K}[x]$ si $f(x)$ no puede escribirse como producto $g(x)h(x)$ de dos polinomios $g(x)$ y $h(x)$ en $\mathbb{K}[x]$, ambos de grado menor que $f(x)$.*

Teorema 2.2.4. *Sea $\mathbb{K}$ un campo, el ideal $\langle p(x)\rangle \neq \{0\}$ de $\mathbb{K}[x]$ es maximal si y solo si $p(x)$ es irreducible sobre $\mathbb{K}$.*

Un teorema muy importante que nos permitirá definir el campo finito de 4 elementos es el siguiente.

Teorema 2.2.5. *Si $p(x)$ es un polinomio irreducible en $\mathbb{K}[x]$, donde $\mathbb{K}$ es un campo, entonces $\mathbb{K}[x]/\langle p(x)\rangle$ es un campo.*

Demostración. Como $p(x)$ es irreducible, por el teorema 2.2.4 $\langle p(x)\rangle$ es maximal, y por el teorema 2.1.1 $\mathbb{K}[x]/\langle p(x)\rangle$ es un campo. □

2.3. Campos finitos

Para esta sección nos interesará trabajar con algunos campos en específico, los campos cuyo número de elementos es finito, es decir, si $\mathbb{K}$ es un campo,

$$|\mathbb{K}| = q,$$

entonces $\mathbb{K}$ es llamado *campo finito*, estos campos también son conocidos como *campos de Galois*.

Definición 2.3.1. *Un subcampo $\mathbb{F}$ es un subconjunto de un campo $\mathbb{K}$ que por sí mismo es un campo.*

Definición 2.3.2. *Un campo $\mathbb{K}$ es un campo de extensión de un campo $\mathbb{F}$ si $\mathbb{F}$ es un subcampo de $\mathbb{K}$.*

Definición 2.3.3. *La característica de un campo se define como el número entero positivo más pequeño p tal que $p \cdot 1 = 0$, o es cero si no existe tal p; aquí $p \cdot 1$ significa p sumandos 1, es decir,*

$$\underbrace{1 + 1 + 1 + \cdots + 1}_{p \text{ veces}} = p \cdot 1.$$

En tal caso se dice que $\mathbb{K}$ es de característica p.

Teorema 2.3.1. *Sea $\mathbb{K}$ un campo finito, de característica p, entonces $p \neq 0$ y además p es primo.*

Demostración. Veamos primero que la característica de $\mathbb{K}$ es distinta de cero, para ello tomemos la identidad en $\mathbb{K}$ y consideremos el siguiente conjunto,

$$\{n \cdot 1 \in \mathbb{K} : n \in \mathbb{N}\}.$$

Como $\mathbb{K}$ es finito existen enteros k y m con $1 \leqslant k < m$ tales que $k \cdot 1 = m \cdot 1$ o equivalentemente $(k - m) \cdot 1 = 0$; por tanto como $k < m$, $k - m \neq 0$, es decir, la característica del campo existe, así, $p \neq 0$.

Ahora como $p \neq 0$, veamos que también es primo, para ello supongamos que no lo es, entonces puede factorizarse como $p = p_1 p_2$ con $p_1, p_2 < p$. Entonces $(p_1 \cdot 1)(p_2 \cdot 1) = p_1 p_2 \cdot 1 = 0$, como $\mathbb{K}$ es un campo, no tiene divisores de cero, es decir, $p_1 \cdot 1 = 0$ ó $p_2 \cdot 1 = 0$, en contradicción de que p es el menor entero positivo tal que cumple esta condición.

Por lo tanto p es un número primo. □

Teorema 2.3.2. *Si $\mathbb{K}$ es un campo finito de característica p, entonces existe un subcampo de $\mathbb{K}$ que es isomorfo a $\mathbb{Z}_p$, el campo de los enteros módulo p.*

Demostración. Consideremos la función,

$$\phi : \mathbb{Z} \to \mathbb{K}$$

dada por,

$$\phi(n) = n \cdot 1$$

para $n \in \mathbb{Z}$.

Veamos que ϕ es un homomorfismo, para ello tomemos $n, m \in \mathbb{Z}$ cualesquiera,

$$\phi(n+m) = (n+m)\cdot 1 = (n\cdot 1) + (m \cdot 1) = \phi(n) + \phi(m).$$

Además la ley distributiva en $\mathbb{K}$ muestra que,

$$\underbrace{(1+1+\cdots+1)}_{n\ sumandos}\underbrace{(1+1+\cdots+1)}_{m\ sumandos} = \underbrace{(1+1+\cdots+1)}_{nm\ sumandos}$$

Así, $(n\cdot 1)(m\cdot 1) = (nm)\cdot 1$, para $n, m \in \mathbb{Z}$, es decir, $\phi(nm) = \phi(n)\phi(m)$.

Por lo tanto, ϕ es un homomorfismo, y el kernel de éste debe ser un ideal en $\mathbb{Z}$, y sabemos que todos los ideales en $\mathbb{Z}$ son de la forma $s\mathbb{Z}$ para alguna $s \in \mathbb{Z}$.

Notemos que de hecho ocurre que $ker(\phi) = p\mathbb{Z}$.

Tomemos primero $m \in ker(\phi)$, entonces $\phi(m) = 0$, es decir, $m \cdot 1 = 0$. Aplicando el algoritmo de la división a m y p se tiene que $m = pq + r$, con $q \in \mathbb{Z}$ y $0 \leq r < p$.

Supongamos que $r > 0$, como $m\cdot 1 = 0$, se tiene que $(pq+r)\cdot 1 = 0$, de aquí que $(p\cdot 1)(q\cdot 1)+r\cdot 1 = 0$ pero como $p\cdot 1 = 0$, obtenemos que $r\cdot 1 = 0$ con $r < p$, pero esto no ocurre pues p es la característica de $\mathbb{K}$. Por lo tanto $r = 0$, así, $m = pq$ con $q \in \mathbb{Z}$, es decir $m \in p\mathbb{Z}$, y con ello queda demostrada una inclusión.

Ahora tomemos $m \in p\mathbb{Z}$, entonces existe $q \in \mathbb{Z}$ tal que $m = pq$, aplicando ϕ tenemos $\phi(m) = \phi(p)\phi(q)$, pero como $\phi(p) = 0$, entonces $\phi(m) = 0$, es decir, $m \in ker(\phi)$.

Con todo esto, se tiene que $ker(\phi) = p\mathbb{Z}$, y por el primer teorema de isomorfismos, $Im(\phi) \simeq \mathbb{Z}/p\mathbb{Z} \simeq \mathbb{Z}_p$, y como $\mathbb{Z}_p$ es un campo pues p es primo , entonces $Im(\phi)$ es también un campo al que denotaremos por $\mathbb{F}$.

Por lo tanto existe $\mathbb{F}$ subcampo de $\mathbb{K}$ que es isomorfo a $\mathbb{Z}_p$. □

2.4. Construcción de códigos a partir de un espacio vectorial

Para codificar y decodificar de manera más práctica y eficiente es útil dotar al alfabeto $\mathcal{A}$ de una estructura algebraica. Fijamos el alfabeto $\mathcal{A} = \mathbb{F}_q$, el campo finito de q elementos. El conjunto de cadenas de longitud n, $\mathcal{A}^n$ es un espacio vectorial sobre $\mathbb{F}_q$ de dimensión n, el cual puede ser identificado como:

$$\mathbb{F}_q^n = \{(x_0, x_1, \ldots, x_{n-1}) : x_i \in \mathbb{F}_q, 0 \leq i \leq n-1\}$$

mediante la asignación $x_0x_1\ldots x_{n-1} \to (x_0, x_1, \ldots, x_{n-1})$. Con esta asignación, definimos la suma de las cadenas $a_0a_1\ldots a_{n-1} + b_0b_1\ldots b_{n-1}$ como

$$(a_0, a_1, \ldots, a_{n-1}) + (b_0, b_1, \ldots, b_{n-1}),$$

y la multiplicación por escalar $\alpha(a_0a_1\dots a_{n-1})$, con $\alpha \in \mathbb{F}_q$, como

$$\alpha((a_0, a_1, \dots, a_{n-1})).$$

Con esto podemos definir los códigos aditivos y los códigos cíclicos como sigue:

Definición 2.4.1. *Un código aditivo C sobre el campo finito $\mathbb{F}_q$, es un subgrupo aditivo de $\mathbb{F}_q^n$. Es decir, para cada par de palabras código, $a_1\dots a_n$, y $b_1\dots b_n \in C$, su suma es una palabra código de C.*

Definición 2.4.2. *Un código C es lineal si es un subespacio de $\mathbb{F}_q^n$. Un código lineal de longitud n, dimensión k, y distancia d es llamado un $[n, k, d]$ código.*

Ejemplo. *Sea $C = \{011, 101, 000, 110\}$ un código sobre $\mathbb{Z}_3$, este código no es lineal, puesto que $2(011) = 022$ no está en C, sin embargo C sí es aditivo sobre $\mathbb{Z}_2$.*

En este trabajo serán de suma importancia los códigos cíclicos pues se espera que mejoren los algoritmos de ADN, y cuya definición es la siguiente.

Definición 2.4.3. *Un código C es cíclico si siempre que la palabra $c_0c_1\dots c_{n-1} \in C$, entonces $c_{n-1}c_0\dots c_{n-2} \in C$.*

Ejemplo. *Sea $C_1 = \{011, 101, 110\}$ un código sobre el campo $\mathbb{Z}_2$, el cual no es un código aditivo ni lineal puesto que 000 no está en C_1, sin embargo si es un código cíclico. Y por otro lado el código $C_2 = \{000000, 000111, 111000, 111111\}$ sobre $\mathbb{Z}_2$ es un código lineal pero no es cíclico.*

2.5. Construcción de códigos cíclicos a partir de un ideal

Para el desarrollo del siguiente capítulo, utilizaremos un anillo en especial, el cual describimos a continuación.

Nota. *Sea $\mathbb{F}$ un campo, consideremos el anillo $\mathbb{F}[x]/\langle x^n - 1\rangle$. A este anillo lo denotaremos $\mathbb{F}^{(n)}[x]$, y a sus elementos los denotaremos simplemente $c(x)$ en lugar de $[c(x)]$.*

Tomemos un elemento $c(x) \in \mathbb{F}^{(n)}[x]$, como ya vimos en el Teorema 2.2.3, todo elemento de $\mathbb{F}^{(n)}[x]$ puede ser representado únicamente por un polinomio de grado $n-1$, por tanto, podemos suponer que $c(x)$ es de grado $n-1$, al multiplicarlo por x, obtenemos

$$xc(x) = c_0x + c_1x^2 + \cdots + c_{n-2}x^{n-1} + c_{n-1}x^n,$$

es decir,

$$xc(x) = c_{n-1} + c_0x + \cdots + c_{n-2}x^{n-1},$$

puesto que $x^n = 1$ en $\mathbb{F}^{(n)}[x]$, esto genera lo que se llama **desplazamiento cíclico**.

Además este anillo tan particular es también un espacio vectorial sobre el campo $\mathbb{F}$, y la multiplicación por escalar es como sigue; sea $a \in \mathbb{F}$ y $c(x) = c_0 + c_1x + \cdots + c_{n-1}x^{n-1} \in \mathbb{F}^{(n)}[x]$, entonces,

$$ac(x) = ac_0 + ac_1x + \cdots + ac_{n-1}x^{n-1}.$$

Ahora construiremos códigos cíclicos lineales a partir de un ideal en el anillo de polinomios, de la siguiente forma:

Los códigos cíclicos lineales pueden definirse a partir de un ideal, para lo cual necesitamos establecer una correspondencia entre los códigos sobre un campo finito y el anillo de polinomios. Para esto consideremos las siguientes funciones,

$$\varphi : \mathbb{F}_q^n \to \mathbb{F}[x]$$

dada por,

$$\varphi(c_0, \ldots, c_{n-1}) = c_0 + c_1x + \cdots + c_{n-1}x^{n-1}$$

y además,

$$\pi : \mathbb{F}[x] \to \mathbb{F}^{(n)}[x],$$

dada por,

$$\pi(p(x)) = [p(x)]$$

donde $[p(x)]$ representa la clase de equivalencia del polinomio $p(x)$.

Teorema 2.5.1. *La función:*

$$\pi \circ \varphi : \mathbb{F}_q^n \to \mathbb{F}^{(n)}[x],$$

es una trasformación lineal. Si C es un código cíclico y lineal, $(\pi \circ \varphi)(C)$ es un ideal. Más aún $\pi \circ \varphi \mid_C : C \to (\pi \circ \varphi)(C)$ es un isomorfismo de espacios vectoriales.

Demostración. Sea C un código lineal cíclico, tomemos $c = c_0c_1 \ldots c_{n-1} \in C$ y $[x] \in \mathbb{F}^{(n)}[x]$, entonces,

$$\begin{aligned}[x][c_0 + c_1x + \cdots + c_{n-1}x^{n-1}] &= [c_0x + c_1x^2 + \cdots + c_{n-1}x^n] \\ &= [c_{n-1} + c_0x + \cdots + c_{n-2}x^{n-1}] \in (\pi \circ \varphi)(C).\end{aligned}$$

esta igualdad es cierta puesto que en $\mathbb{F}^{(n)}[x]$, $x^n = 1$. Análogamente se prueba que $[c_0 + c_1x + \cdots + c_{n-1}x^{n-1}][x] \in \mathbb{F}^{(n)}[x]$. Por lo tanto, $(\pi \circ \varphi)(C)$ es un ideal bilateral.

Veamos que la función $\pi \circ \varphi : \mathbb{F}_q^n \to \mathbb{F}^{(n)}[x]$, es una transformación lineal, para ello, sean

$(a_0, a_1, \ldots, a_{n-1}), (b_0, b_1, \ldots, b_{n-1}) \in \mathbb{F}_q^n$, y $\alpha \in \mathbb{F}$, entonces,

$$
\begin{aligned}
&(\pi \circ \varphi)(\alpha(a_0, a_1, \ldots, a_{n-1}) + (b_0, b_1, \ldots, b_{n-1})) = \\
&(\pi \circ \varphi)((\alpha a_0 + b_0, \alpha a_1 + b_1, \ldots, \alpha a_{n-1} + b_{n-1}) = \\
&[(\alpha a_0 + b_0) + (\alpha a_1 + b_1)x + \cdots + (\alpha a_{n-1} + b_{n-1})x^{n-1}] = \\
&[\alpha a_0 + \alpha a_1 x + \cdots + \alpha a_{n-1}x^{n-1}] + [b_0 + b_1 x + \cdots + b_{n-1}x^{n-1}] = \\
&\alpha[a_0 + a_1 x + \cdots + a_{n-1}x^{n-1}] + [b_0 + b_1 x + \cdots + b_{n-1}x^{n-1}] = \\
&\alpha(\pi \circ \varphi)(a_0, a_1, \ldots, a_{n-1}) + (\pi \circ \varphi)(b_0, b_1, \ldots, b_{n-1}).
\end{aligned}
$$

Por lo tanto $\pi \circ \varphi$ es una transformación lineal.

Como la restricción está definida en C se tiene de inmediato que $\pi \circ \varphi \mid_C \colon C \to (\pi \circ \varphi)(C)$ es sobre.

Para ver que $\pi \circ \varphi \mid_C \colon C \to (\pi \circ \varphi)(C)$ es inyectiva, por la linealidad de la función, basta ver que $ker(\pi \circ \varphi \mid_C) = \{0\}$, tomemos $c = c_0 c_1 \ldots c_{n-1} \in ker(\pi \circ \varphi \mid_C)$ entonces $(\pi \circ \varphi \mid_C)(c) = [c_0 + c_1 x + \cdots + c_{n-1}x^{n-1}] = [0]$, como ya vimos que la restricción es sobre, existen palabras código $c_0 c_1 \ldots c_{n-1}$ y $00\ldots0$ tales que $c_0 c_1 \ldots c_{n-1} = 00\ldots0$, esto implica que $c_i = 0$ para todo $i \in \{0, 1, \ldots, n-1\}$, por lo tanto la restricción es inyectiva.
Con todo esto afirmamos que $\pi \circ \varphi \mid_C \colon C \to (\pi \circ \varphi)(C)$ es biyectiva, por tanto es un isomorfismo de espacios vectoriales. □

Por el resultado anterior, podemos trabajar indistintamente con elementos en C o con la correspondiente clase de equivalencia $c(x)$.

En general, si $p(x)$ es irreducible sobre el campo $\mathbb{F}$ y $grad(p(x)) = n$, entonces por el Teorema 2.2.3 a los elementos de $\mathbb{F}[x]/\langle p(x)\rangle$ los identificaremos con $a_0 + a_1 x + \cdots + a_{n-1}x^{n-1}$.

Teorema 2.5.2. *Sea J un ideal no cero de $\mathbb{F}^{(n)}[x]$. El conjunto*

$$\{(a_0, \ldots, a_{n-1}) \in \mathbb{F}^n : [a_0 + a_1 x + \cdots + a_{n-1}x^{n-1}] \in J\}$$

es un código lineal cíclico.

Demostración. Llamemos C al conjunto $\{(a_0, \ldots, a_{n-1}) \in \mathbb{F}^n : a_0 + a_1 x + \cdots + a_{n-1}x^{n-1} \in J\}$. Veamos primero que C es un espacio vectorial, tomemos $(a_0, a_1, \ldots a_{n-1})$ y $(b_0, b_1, \ldots, b_{n-1}) \in \mathbb{F}^n$, entonces $a_0 + a_1 x + \cdots + a_{n-1}x^{n-1} \in J$ y $b_0 + b_1 x + \cdots + b_{n-1}x^{n-1} \in J$, como J es un ideal de $\mathbb{F}^{(n)}[x]$, entonces $a_0 + a_1 x + \cdots + a_{n-1}x^{n-1} + b_0 + b_1 x + \cdots + b_{n-1}x^{n-1} \in J$, es decir, $(a_0+b_0)+(a_1+b_1)x+\cdots+(a_{n-1}+b_{n-1})x^{n-1} \in J$, por lo tanto $(a_0+b_0, a_1+b_1, \ldots, a_{n-1}+b_{n-1})] \in C$.

Además si tomamos $\alpha \in \mathbb{F}^n$, entonces, como J es un ideal,

$$\alpha(a_0 + a_1 x + \cdots + a_{n-1}x^{n-1}) = \alpha a_0 + \alpha a_1 x + \cdots + \alpha a_{n-1}x^{n-1} \in J,$$

entonces $(\alpha a_0, \alpha a_1, \ldots, \alpha a_{n-1}) \in C$.

Por lo tanto C es un código lineal.

Ahora veamos que C es cíclico, para ello tomemos $(c_0, c_1, \ldots, c_{n-1}) \in C$ cualquiera, entonces, $c_0 + c_1x + \cdots + c_{n-1}x^{n-1} \in J$, y sea $x \in \mathbb{F}^{(n)}[x]$, luego

$$\begin{aligned} x(c_0 + c_1x + \cdots + c_{n-1}x^{n-1}) &= c_0x + c_1x^2 + \cdots + c_{n-1}x^n \\ &= c_{n-1} + c_0x + c_1x^2 + \cdots + c_{n-2}x^{n-1} \in J \end{aligned}$$

pues J es un ideal.

Por lo tanto $(c_{n-1}, c_0, \cdots, c_{n-2}) \in C$, es decir, C es cíclico. □

Teorema 2.5.3. *Sea J un ideal no cero en $\mathbb{F}^{(n)}[x]$. Existe un único polinomio mónico $g(x) \in \mathbb{F}[x]$ de grado r tal que,*

1. *$J = \langle g(x)\rangle / \langle x^n - 1\rangle$.*

2. *$g(x)$ divide a $x^n - 1$.*

Además, si C es el código cíclico lineal correspondiente a J, entonces la dimensión de C es $n - r$.

Demostración. 1. Sea J un ideal no cero de $\mathbb{F}^{(n)}[x]$, por el Teorema 2.1.3, a J le corresponde un ideal J' de $\mathbb{F}[x]$, tal que $J = J'/\langle x^n - 1\rangle$ y como $\mathbb{F}$ es un campo, J' es un ideal principal, esto es, $J' = \langle g(x)\rangle$.

Veamos ahora que este polinomio $g(x)$ es el único polinomio mónico en J' con grado mínimo r.

Sean $f(x) = f_0 + f_1x + \cdots + f_{r-1}x^{r-1} + x^r$, $g(x) = g_0 + g_1x + \cdots + g_{r-1}x^{r-1} + x^r \in J'$ polinomios mónicos distintos de grado mínimo r. Luego, $f(x) - g(x) = (f_0 - g_0) + (f_1 - g_1)x + \cdots + (f_{r-1} - g_{r-1})x^{r-1} \in J'$, pues J' es un ideal, pero $f(x) - g(x)$ es un polinomio que tiene menor grado que $f(x)$, y $g(x)$, lo cual no puede ocurrir, pues ambos son de grado mínimo, por lo tanto, el polinomio mónico $g(x)$ es el único de grado mínimo en J'.

2. Aplicando el algoritmo de la división en $\mathbb{F}[x]$, tenemos $x^n - 1 = h(x)g(x) + r(x)$ en $\mathbb{F}[x]$, donde $grad(r(x)) < r$, de aquí que

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

es decir, $(x^n - 1) - r(x) \in J' = \langle g(x)\rangle$ y como J' es ideal, $r(x) \in J'$, pero como $grad(r(x)) < r$, esto no puede ocurrir a menos que $r(x) = 0$.

Por lo tanto, $x^n - 1 = h(x)g(x)$, esto es, $g(x)$ divide a $x^n - 1$.

Ahora tomemos $c(x) \in J'$, el $grad(c(x)) < n$, y $c(x) = q(x)g(x) \in \mathbb{F}[x]$, entonces,

$$\begin{aligned} c(x) &= q(x)g(x) + e(x)(x^n - 1) \in \mathbb{F}[x] \\ &= (q(x) + e(x)h(x))g(x) \in \mathbb{F}[x] \\ &= f(x)g(x) \in \mathbb{F}[x], \text{ donde } f(x) = q(x) + e(x)h(x), \end{aligned}$$

Ahora veamos cómo es el grado de $f(x)$. Notemos que como $c(x) = f(x)g(x)$, entonces $grad(c(x)) = grad(f(x)) + r$, es decir, $grad(f(x)) = grad(c(x)) - r < n - r$.

Así J' consiste de múltiplos de $g(x)$ por polinomios de grado menor o igual que $n - r - 1$ evaluados en $\mathbb{F}[x]$.

Luego, para ver que la dimensión de $C = \{(a_0, a_1, \ldots, a_{n-1}) \in \mathbb{F}^n : a_0 + a_1x + \cdots + a_{n-1}x^{n-1} \in J\}$ es $n - r$ basta ver que la dimensión de J es también $n - r$, o bien hay que ver que $dim(J')$ es $n - r$, para ello notemos que hay $n - r$ múltiplos linealmente independientes de $g(x)$. Esto es claro, pues los múltiplos de $g(x)$ son, $g(x), xg(x), x^2g(x), \ldots, x^{n-r-1}g(x)$, los cuales son linealmente independientes. Además este conjunto de múltiplos de $g(x)$ genera a J'. Esto es fácil de ver tomando $c(x) \in J'$, esto es,

$$\begin{aligned} c(x) &= g(x)h(x) \\ &= g(x)(h_0 + h_1x + \cdots + h_{n-r-1}x^{n-r-1}) \\ &= h_0g(x) + h_1(xg(x)) + \cdots + h_{n-r-1}(x^{n-r-1}g(x)). \end{aligned}$$

Por lo tanto el conjunto $\{g(x), xg(x), \ldots, x^{n-r-1}g(x)\}$ es una base para J' y tiene $n - r$ vectores, así la dimensión de J' es $k = n - r$, y como $J = J'/\langle x^n - 1\rangle$, entonces $dim(J) = n - r$, y por consiguiente la dimensión de C, el código asociado a J es también $n - r$.

□

Capítulo 3

Códigos de ADN y códigos finitos

3.1. Códigos de ADN

Una hebra simple de ADN es una sucesión de cuatro posibles nucleótidos: Adenina (A), Guanina (G), Citosina (C) y Timina (T). Los extremos de una hebra de ADN son químicamente polares y son llamados extremos $5'$ y $3'$. Las aplicaciones del ADN requieren éxito en la hibridación específica entre una palabra código de ADN y su complemento Watson-Crick. El complemento Watson-Crick de una hebra de ADN se obtiene al reemplazar cada A por T y viceversa, cada C por G y viceversa, e intercambiar los extremos $3'$ y $5'$. Por ejemplo, el complemento Watson-Crick de la hebra $3' - CCATTGA - 5'$ es $5' - GGTAACT - 3'$.

Con el proceso de hibridación específica entre una hebra de ADN y su complemento Watson-Crick se forma la doble hélice de la cadena de ADN, como puede verse en la Figura 3.1. El ADN ocurre en secuencias, representadas por secuencias de letras del alfabeto $\{A, C, G, T\}$. El complemento Watson-Crick está dado por $A^c = T, T^c = A, C^c = G$, y $G^c = C$.

3.2. Construcción del campo $GF(4)$

En esta sección explicaremos cómo se construye el campo $GF(4)$, el cual es un campo de Galois, o campo finito de 4 elementos. Para esto, tomaremos como base el campo de los números binarios $(GF(2), +, *)$, donde $GF(2) = \{0, 1\}$ es el anillo de los enteros módulo 2, con las operaciones de suma "+" y producto "$*$" usuales en este anillo.

Sea $GF(2)[x]$, el anillo de los polinomios en una variable con coeficientes en $GF(2)$. Tomemos el polinomio $p(x) = x^2 + x + 1 \in GF(2)[x]$, el cual sabemos que es irreducible en $GF(2)$, ya que es de grado 2 y que no tiene raíces en el campo. Ahora consideremos el anillo cociente $GF(2)[x]/\langle p(x)\rangle$,

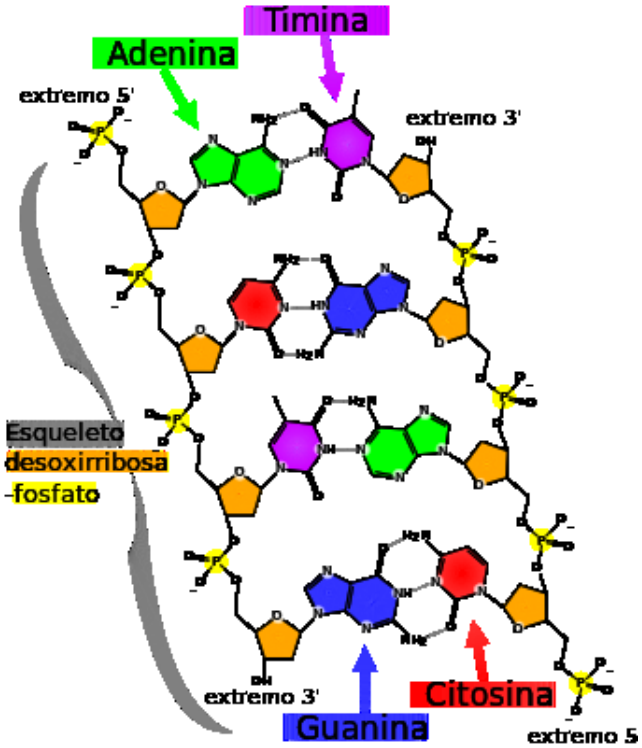


Figura 3.1: Doble hélice de la cadena de ADN

que resulta ser un campo, donde $\langle p(x)\rangle$ es el ideal generado por el polinomio $p(x)$. En este caso,

$$GF(2)/\langle p(x)\rangle = \{a(x) + \langle p(x)\rangle : a(x) \in GF(2)[x]\}.$$

Como $GF(2)[x]$ es un anillo euclidiano, usando el algoritmo de la división, se tiene que para cualquier elemento $a(x) \in GF(2)[x]$,

$$a(x) = p(x)q(x) + r(x), \text{ con } 0 \leq grad(r(x)) \leq 1.$$

Por lo tanto, un representante de la clase de $a(x)$, dada por $[a(x)] = a(x) + \langle p(x)\rangle$ es: $a_0 + a_1x$, con $a_0, a_1 \in GF(2)$.

Por consiguiente, los elementos del anillo $GF(2)/\langle p(x)\rangle$ se pueden identificar mediante el conjunto $GF(4)$, además, por el hecho de ser $p(x)$ irreducible, $\langle p(x)\rangle$ es maximal, y por el teorema 2.2.5, $GF(4)$ es un campo.

$$GF(4) = \{a_0 + a_1x : a_0, a_1 \in GF(2)\} = \{0, 1, \omega, 1+\omega\} = \{0, 1, \omega, \overline{\omega}\},$$

donde $\omega = [x] = x + \langle p(x)\rangle$, y $\overline{\omega} = 1 + \omega$, además $\overline{\omega} = \omega^2$, y $1 + \omega + \omega^2 = 0$.

3.3. Códigos asociados con el ADN

Las cadenas de ADN tienen características muy especiales que resultan importantes para el desarrollo de los algoritmos, una de estas características es que cada hebra de ADN tiene asociada una hebra complementaria, que es su complemento Watson-Crick.

En este trabajo, asociaremos códigos sobre el conjunto $\{A, C, G, T\}$ con códigos sobre $GF(4) = \{0, 1, \omega, \overline{\omega}\}$, es decir asociaremos 0 con Adenina(A), ω con Citosina(C), $\overline{\omega}$ con Guanina(G) y 1 con Timina(T). Para definir el complemento de una palabra código en $GF(4)$, definiremos primero el complemento de un elemento de $GF(4)$.

Definición 3.3.1. *Sea $a \in GF(4)$, el complemento de a, denotado por a^c, se define como $a^c = a+1$.*

Con esto, se tiene la siguiente definición.

Definición 3.3.2. *El complemento de una palabra $u = u_0 \dots u_{n-1}$ en $GF(4)$ es $u^c = {u_0}^c \dots {u_{n-1}}^c$, y el reverso de u es $u^r = u_{n-1} \dots u_0$.*

Notemos que si damos una asociación diferente a la que presentamos, del conjunto de nucleótidos con el campo $GF(4)$, no se tienen las definiciones anteriores.

Ejemplo. *El complemento de la palabra $x = 010\overline{\omega}\omega$ es $x^c = 101\omega\overline{\omega}$ y su reverso es $x^r = \omega\overline{\omega}010$.*

Los códigos que tienen estas propiedades se definen como sigue,

Definición 3.3.3. *Un código aditivo C de longitud n sobre $GF(4)$ es llamado reversible si $u^r \in C$, para todo $u \in C$.*

Ejemplo. *El código $C = \{110, 011, 101, 000\}$ es reversible pero el código $C' = \{111, 100, 011, 000\}$ no lo es.*

El reverso de una palabra código tiene un equivalente en el anillo de polinomos, esto es lo que se llama recíproco de un polinomio.

Definición 3.3.4. *Para cada polinomio $p(x) = p_0 + p_1x + \cdots + p_rx^r$ con $p_r \neq 0$, definimos el recíproco de $p(x)$ como el polinomio $p^*(x) = x^r p(1/x) = p_r + p_{r-1}x + \cdots + p_0x^r$. Además diremos que un polinomio es auto-recíproco si y sólo si $p(x) = p^*(x)$.*

Definición 3.3.5. *Un código aditivo C de longitud n sobre $GF(4)$ es llamado complemento si $u^c \in C$, para todo $u \in C$.*

Definición 3.3.6. *Un código aditivo C sobre $GF(4)$ es llamado complemento cíclico reversible si,*

1. *C es cíclico.*

2. *C es reversible.*

3. C es complemento.

3.4. Resultados sobre códigos cíclicos y aditivos

En esta sección daremos algunos resultados que sólo ocurren en el campo $GF(4)$.

En adelante consideraremos a los elementos de los códigos cíclicos como polinomios, puesto que ya vimos en el capítulo anterior que hay una correspondencia entre ellos y los ideales en $\mathbb{F}^{(n)}[x]$, también será necesario definir el grado de un polinomio en un código cíclico como sigue,

Definición 3.4.1. *Sea C un código cíclico lineal correspondiente a $J = \langle g(x)\rangle/\langle x^n - 1\rangle$ un ideal de $GF(4)^{(n)}[x]$, a su vez por el Teorema 2.1.3, a J le corresponde un ideal J' de $GF(4)[x]$, tal que $J = J'/\langle x^n - 1\rangle$. Como cada elemento de J es una clase de equivalencia, definimos el grado de $[f(x)] \in J$ como el grado del polinomio $f(x) \in J'$.*

Así, el grado de $f(x) \in C$ será el grado de $[f(x)] \in J$, y lo denotaremos como $grad(f(x))$.

El siguiente teorema es un caso particular del teorema 2.5.3 utilizando códigos sobre el campo $GF(4)$.

Teorema 3.4.1. *Sea C un $[n, k, d]$-código cíclico lineal de longitud n sobre $GF(4)$. Entonces $C = \langle g(x)\rangle$, donde $g(x)$ es un polinomio mónico de grado r que divide a $x^n - 1$ sobre $GF(4)$, y $k = n - r$.*

Ahora veamos un teorema que muestra cómo son los polinomios generadores de un código cíclico aditivo, para ello, consideremos la función *traza*,

$$Tr : GF(4) \to GF(2) \text{ definida por } Tr(x) = x + x^2 \tag{3.1}$$

En particular $Tr(0) = Tr(1) = 0$, $Tr(\omega) = \omega + \omega^2 = 1$, puesto que $1 + \omega + \omega^2 = 0$, $\overline{\omega} = \omega + 1$, y $Tr(\overline{\omega}) = 1$.

Teorema 3.4.2. *La función traza es un homomorfismo de grupos aditivos.*

Demostración. Sean $x, y \in GF(4)$, entonces,

$$Tr(x + y) = (x + y) + (x + y)^2 = x + y + x^2 + xy + yx + y^2.$$

Note que en $GF(4)$ cuando sumamos un elemento con él mismo, ocurre que la suma da cero, es decir $GF(4)$ es de característica 2. Así que $xy + yx = xy + xy = 0$. Por lo tanto,

$$Tr(x + y) = x + y + x^2 + y^2 = (x + x^2) + (y + y^2) = Tr(x) + Tr(y).$$

Es decir, la función $Tr : GF(4) \to GF(2)$ es un homomorfismo de grupos aditivos. □

Teorema 3.4.3. *Sea C un $(n, 2^k)$-código cíclico aditivo de longitud n sobre $GF(4)$. Entonces $C = \langle \omega p(x) + q(x), r(x) \rangle$, donde $p(x)$ y $r(x)$ son polinomios binarios que dividen a $(x^n - 1)$ (mod 2), $r(x)$ divide a $\frac{q(x)(x^n-1)}{p(x)}$ (mod 2), y $k = 2n - grad(p(x)) - grad(r(x))$.*

Demostración. Consideremos la función $T : C \to \mathbb{Z}_2^{(n)}[x]$ obtenida al aplicar la función traza (3.1) a cada elemento de cada palabra del código C. Puesto que C es un código aditivo pero no sabemos si es lineal, por el teorema 3.4.2 sólo podemos afirmar que para todo $c(x), b(x) \in C$

$$T(c(x) + b(x)) = T(c(x)) + T(b(x)).$$

Primero mostremos que el kernel de esta función es un código binario cíclico aditivo, para ello, notemos que para que un elemento $f(x) \in C$ esté en $ker(T)$, sus coeficientes deben ser 0 ó 1, pues $Tr(0) = Tr(1) = 0$, es decir, podemos considerar los polinomios del $ker(T)$ como elementos de $\mathbb{Z}_2^{(n)}[x]$. Por tanto, veamos que $ker(T)$ es un ideal de $\mathbb{Z}_2^{(n)}[x]$, es decir, hay que ver que para $a(x), b(x) \in ker(T)$, se tiene que $a(x) + b(x) \in ker(T)$, lo cual ocurre debido a que la función traza distribuye la suma. Además si tomamos $a(x) \in ker(T)$ y $c(x) \in \mathbb{Z}_2^{(n)}[x]$, como ambos son binarios, entonces $m(x) = a(x)c(x)$ es también binario; además $c(x) \in GF(4)^{(n)}[x]$ y se tiene que C es un código cíclico sobre $GF(4)$, por lo cual podemos considerarlo como un ideal de $GF(4)^{(n)}[x]$ y por lo tanto podemos aplicar la función T a $m(x)$, es decir, $T(m(x)) = 0$.

Con todo esto, $ker(T)$ es un ideal de $\mathbb{Z}_2^{(n)}[x]$, y por el teorema 2.5.3, $ker(T)$ es un código cíclico lineal, además por el teorema anterior, $ker(T) = \langle r(x) \rangle$, y $r(x)$ es un factor de $x^n - 1$. Además $dim(ker(T)) = n - grad(r(x))$.

Ahora hay que ver también que la imagen de C es un ideal de $\mathbb{Z}_2^{(n)}[x]$. Para ello, tomemos $a(x), b(x) \in T(C)$, es decir, polinomios binarios obtenidos de algunos polinomios $A(x),\ B(x) \in C$ al aplicarles la función T respectivamente, esto es

$$a(x) + b(x) = T(A(x)) + T(B(x)) = T(A(x) + B(x))$$

Por lo tanto, $a(x) + b(x) \in T(C)$, esto es $T(C)$ es un subgrupo aditivo de $\mathbb{Z}_2^{(n)}[x]$. Además sean $a(x) \in T(C)$ y $b(x) \in \mathbb{Z}_2^{(n)}[x]$, entonces existe $A(x) \in C$ tal que $T(A(x)) = a(x)$, y como $b(x)$ es

binario, entonces los coeficientes de $a(x)b(x) = T(A(x))b(x)$ son de la siguiente forma

$$\begin{aligned}
a(x)b(x) &= (Tr(A_0) + Tr(A_1)x + \cdots + Tr(A_{n-1})x^{n-1})(b_0 + b_1x + \cdots + b_{n-1}x^{n-1}) \\
&= (Tr(A_0) + Tr(A_1)x + \cdots + Tr(A_{n-1})x^{n-1})b_0 \\
&+ (Tr(A_0) + Tr(A_1)x + \cdots + Tr(A_{n-1})x^{n-1})b_1x \\
&+ \cdots + (Tr(A_0) + Tr(A_1)x + \cdots + Tr(A_{n-1})x^{n-1})b_{n-1}x^{n-1} \\
&= (Tr(A_0) + Tr(A_1)x + \cdots + Tr(A_{n-1})x^{n-1})b_0 \\
&+ (Tr(A_0)x + Tr(A_1)x^2 + \cdots + Tr(A_{n-2})x^{n-1} + Tr(A_{n-1})b_1 \\
&+ \cdots + (Tr(A_0)x^{n-1} + Tr(A_1) + \cdots + Tr(A_{n-2})x^{n-3} + Tr(A_{n-1})x^{n-2})b_{n-1} \\
&= (Tr(A_0)b_0 + Tr(A_{n-1})b_1 + \cdots + Tr(A_1)b_{n-1} \\
&+ ((Tr(A_1)b_0 + Tr(A_0)b_1 + \cdots + Tr(A_2)b_{n-1})x \\
&+ \cdots + (Tr(A_{n-1})b_0 + Tr(A_{n-2})b_1 + \cdots + Tr(A_0)b_{n-1})x^{n-1}
\end{aligned}$$

Como los coeficientes de $b(x)$ son 0 ó 1, se tiene que los coeficientes del producto anterior son sumas de coeficientes de A con la función Tr, la cual distribuye la suma, por lo tanto los coeficientes del producto son de la forma $Tr(A'_i)$, donde para cada $i \in \{1, 2, \ldots, n-1\}$, A'_i es una suma de algunos $A_j,\ j \in \{1, 2, \ldots, n-1\}$.

Con todo esto, existe un polinomio $A'(x) \in C$, puesto que puede verse como suma de elementos de C, el cual es un código aditivo; además $T(A'(x)) = a(x)b(x)$. Así, $T(C)$ es un ideal de $\mathbb{Z}_2^{(n)}[x]$, y por el teorema 2.5.3, $T(C)$ es un código cíclico lineal, y es generado por un polinomio $p(x)$.

Ahora veamos quién es el generador de los polinomios en C que no están en el kernel, para ello tomemos al polinomio $p(x)$, el cual está en $T(C)$, por tanto, existe un polinomio $s(x) \in C$ tal que

$$T(s(x)) = p(x),$$

entonces

$$s(x) = s_0 + s_1x + \cdots + s_{n-1}x^{n-1}$$

y

$$p(x) = T(s(x)) = T(s_0) + T(s_1)x + \cdots + T(s_{n-1})x^{n-1}.$$

Definamos los siguientes conjuntos,

$$I = \{k : p_k = 1\} = \{k : s_k = \omega \text{ ó } s_k = \overline{\omega}\} = \{k : s_k = \omega\} \cup \{k : s_k = \overline{\omega}\},$$

pongamos $I_1 = \{k : s_k = \omega\}$ e $I_2 = \{k : s_k = \overline{\omega}\}$.

$$J = \{k : p_k = 0\} = \{k : s_k = 0 \text{ ó } s_k = 1\}.$$

Con esto podemos reescribir a $s(x)$ como,

$$\begin{aligned}
s(x) &= \sum_{k\in J} s_k x^k + \sum_{k\in I_1} s_k x^k + \sum_{k\in I_2} s_k x^k \\
&= \sum_{k\in J} s_k x^k + \sum_{k\in I_1} \omega x^k + \sum_{k\in I_2} \overline{\omega} x^k \\
&= \sum_{k\in J} s_k x^k + \omega \sum_{k\in I_1} x^k + \overline{\omega} \sum_{k\in I_2} x^k \\
&= \sum_{k\in J} s_k x^k + \omega \sum_{k\in I_1} x^k + (1+\omega) \sum_{k\in I_2} x^k \\
&= \sum_{k\in J} s_k x^k + \omega(\sum_{k\in I_1\cup I_2} x^k) + \sum_{k\in I_2} x^k \\
&= \sum_{k\in J} s_k x^k + \omega(\sum_{k\in I} x^k) + \sum_{k\in I_2} x^k \\
&= \sum_{k\in J} s_k x^k + \omega p(x) + \sum_{k\in I_2} x^k,
\end{aligned}$$

donde $p(x) = \sum_{k\in I} x^k$.

Así, $s(x)$ es la suma de un polinomio binario, al que llamaremos $q(x)$, más el polinomio $\omega p(x)$, es decir $s(x) = \omega p(x) + q(x)$.

Por lo tanto, el código C es generado por $r(x)$ y alguna imagen inversa de $p(x)$, digamos $\omega p(x) + q(x)$.

Recordemos que $dim(C) = dim(ker(T)) + dim(Im(T))$, sobre $\mathbb{Z}_2$, por el Teorema 2.5.3, $dim(ker(T)) = n - grad(r(x))$ y $dim(Im(T)) = n - grad(p(x))$, así que $dim(C) = 2n - grad(r(x)) - grad(p(x))$.

Finalmente, probemos que $r(x)$ divide a $\frac{q(x)(x^n-1)}{p(x)}$, dado que,

$$\frac{q(x)(x^n-1)}{p(x)} = \frac{(x^n-1)}{p(x)}(\omega p(x) + q(x)) + \omega(x^n-1) \text{ y como } r(x) \text{ divide a } x^n-1,$$

entonces $x^n - 1 = r(x)k(x)$ para algún polinomio $k(x)$, y así,

$$\frac{q(x)(x^n-1)}{p(x)} = \frac{(x^n-1)}{p(x)}(\omega p(x) + q(x)) + r(x)(\omega k(x)) \in C.$$

Como ya vimos, $\omega(x^n-1) = r(x)(\omega k(x))$, es decir, $\omega(x^n-1) \in \langle r(x)\rangle$, por tanto es un polinomio binario.

Por otro lado, $\frac{(x^n-1)}{p(x)}(\omega p(x)+q(x)) = \omega(x^n-1)+q(x)(x^n-1)$, es suma de polinomios binarios, entonces $\frac{(x^n-1)}{p(x)}(\omega p(x) + q(x))$ es binario.

Con todo esto, $\frac{q(x)(x^n-1)}{p(x)}$ es también un polinomio binario, luego $\frac{q(x)(x^n-1)}{p(x)} = r(x)g(x)$, para algún polinomio $g(x)$, es decir, $r(x)$ divide a $\frac{q(x)(x^n-1)}{p(x)}$. □

Nota. *Cuando algún polinomio generador es nulo, se tiene lo siguiente.*

- *Si $p(x) = 0$, entonces $k = n - grad(r(x))$, y si $r(x) = 0$, se tiene que $k = n - grad(p(x))$.*

- *Como para cualquier subgrupo se tiene que* $\langle a, b\rangle = \langle a, b-a\rangle$*, nosotros asumiremos que* $grad(q(x)) < grad(r(x))$.

Ahora, veamos algunos resultados sobre códigos cíclicos reversibles y los polinomios generadores.

Teorema 3.4.4. *Un código lineal* C *cíclico* $\langle g(x)\rangle$ *sobre* $GF(4)$ *es reversible si y solo si* $g(x)$ *es auto-recíproco.*

Demostración. Supongamos primero que C es reversible, para ello consideremos el conjunto de polinomios recíprocos de los polinomios de C, $C' = \{f^*(x) : f(x) \in C\}$, veamos que $C' = \langle g^*(x)\rangle$, entonces tomemos $f^*(x) \in C'$

$$\begin{aligned} f^*(x) &= x^{n-1}f(x^{-1}) \\ &= x^{n-1}g(x^{-1})j(x^{-1}) \\ &= (x^{n-r-1}j(x^{-1}))(x^r g(x^{-1})) \\ &= j^*(x)g^*(x) \end{aligned}$$

Por lo tanto, $f^*(x) \in \langle g^*(x)\rangle$. La otra inclusión es clara. Así $C' = \langle g^*(x)\rangle$. Además como C es reversible, entonces $\langle g(x)\rangle = C = C' = \langle g^*(x)\rangle$.

Así $g^*(x)$ también genera a C y al ser $g(x)$ el polinomio de grado mínimo que genera a C, se tiene que $g(x) = g^*(x)$, es decir, $g(x)$ es auto-recíproco.

Recíprocamente, supongamos que $g(x) = g^*(x) = x^r g\left(\frac{1}{x}\right)$, donde $r = grad(g(x))$. Tomemos $p(x) \in C$, con $grad(p(x)) = m > r$. Debemos mostrar que $p^*(x) \in C$, es decir, $p^*(x)$ es un múltiplo de $g(x)$. Calculemos $p^*(x)$,

$$\begin{aligned} p^*(x) &= x^m p\left(\frac{1}{x}\right) = x^m k\left(\frac{1}{x}\right) g\left(\frac{1}{x}\right) \\ &= x^{r+s}k\left(\frac{1}{x}\right) g\left(\frac{1}{x}\right), \text{ donde } m = r+s, \text{ para algún } s > 0, \\ &= x^s k\left(\frac{1}{x}\right)\left(x^r g\left(\frac{1}{x}\right)\right) = x^s k\left(\frac{1}{x}\right)(g(x)). \end{aligned}$$

Por lo tanto, $p^*(x) \in C$, es decir, C es reversible. □

Enseguida mostraremos un lema que enuncia algunas propiedades útiles del recíproco de un polinomio.

Lema 3.4.1. *Sean* $f(x), g(x)$ *dos polinomios en* $GF(4)[x]$*, con* $grad(f(x)) = m$*, y* $grad(g(x)) = k$*,* $m \geq k$*, entonces,*

1. $[f(x)g(x)]^* = f^*(x)g^*(x)$.
2. $[f(x) + g(x)]^* = f^*(x) + x^{grad(f(x))-grad(g(x))}g^*(x)$.

Demostración. Sean $m = grad(f(x))$ y $k = grad(g(x))$,

1.

$$\begin{aligned}[f(x)g(x)]^* &= x^{m+k} f\left(\frac{1}{x}\right) g\left(\frac{1}{x}\right) \\ &= x^m x^k f\left(\frac{1}{x}\right) g\left(\frac{1}{x}\right) \\ &= x^m f\left(\frac{1}{x}\right) x^k g\left(\frac{1}{x}\right) \\ &= f^*(x)g^*(x).\end{aligned}$$

2.

$$\begin{aligned}[f(x)+g(x)]^* &= x^m (f\left(\frac{1}{x}\right) + g\left(\frac{1}{x}\right)) \\ &= x^m f\left(\frac{1}{x}\right) + x^m g\left(\frac{1}{x}\right) \\ &= f^*(x) + x^m g\left(\frac{1}{x}\right) \\ &= f^*(x) + x^m x^k x^{-k} g\left(\frac{1}{x}\right) \\ &= f^*(x) + x^{m-k} g^*(x).\end{aligned}$$

□

En la siguiente sección abordaremos algunos resultados acerca de los códigos complementos cíclicos reversibles sobre $GF(4)$.

3.5. Códigos complementos cíclicos reversibles sobre $GF(4)$

Iniciamos esta sección con un lema sobre polinomios auto-recíprocos.

Lema 3.5.1. *Sea $x^n - 1 = g_1(x)g_2(x)g_3(x)$, donde $g_1(x), g_2(x)$, y $g_3(x)$ son polinomios no triviales que dividen a $x^n - 1$ en $GF(4)[x]/(x^n - 1)$. Se cumple que,*

1. *Si $g_1(x)$ y $g_2(x)$ son auto-recíprocos, entonces $g_1(x)g_2(x)$ es auto-recíproco.*
2. *Si $g_1(x)$ ó $g_2(x)$, (pero no ambos), no es auto-recíproco, entonces $g_1(x)g_2(x)$ no es auto-recíproco.*
3. *Si $g_1(x)$ y $g_2(x)$ no son auto-recíprocos, entonces $g_1(x)g_2(x)$ podría ser auto- recíproco.*

Demostración. 1. Supongamos que $g_1(x)$ y $g_2(x)$ son auto-recíprocos, entonces, por el Lema 3.4.1,

$$[g_1(x)g_2(x)]^* = g_1^*(x)g_2^*(x) = g_1(x)g_2(x),$$

es decir, $g_1(x)g_2(x)$ es auto-recíproco.

2. Sean $g_1(x)$ un polinomio auto-recíproco y $g_2(x)$ un polinomio que no lo es. Supongamos que $g_1(x)g_2(x)$ es auto-recíproco. Entonces,

$$g_1(x)g_2(x) = [g_1(x)g_2(x)]^* = g_1^*(x)g_2^*(x) = g_1(x)g_2^*(x),$$

luego, $g_1(x)g_2(x) - g_1(x)g_2^*(x) = 0 = x^n - 1$, lo que es una contradicción, pues $x^n - 1 = g_1(x)g_2(x)g_3(x)$ y $g_3(x)$ es no trivial.

3. Bastará con dar ejemplos de que esto ocurre. Primero veamos que el producto de polinomios no auto-recíprocos es no auto-recíproco, tomemos $g_1(x) = x + \omega^2$ y $g_2(x) = x^2 + x + \omega$, dos divisores de $x^{15} - 1$ sobre $GF(4)$. Dado que,

$$g_1^*(x) = xg_1\left(\frac{1}{x}\right) = 1 + x\omega^2 \neq g_1(x),$$

y

$$g_2^*(x) = x^2 g_2\left(\frac{1}{x}\right) = 1 + x + x^2\omega \neq g_2(x),$$

tenemos que $g_1(x)$ y $g_2(x)$ no son auto-recíprocos. Luego, $g_1(x)g_2(x) = x^3 + x + \omega x^2 + 1$ y $g_1^*(x)g_2^*(x) = x^3 + x\omega + x^2 + 1$, es decir $g_1(x)g_2(x)$ no es auto-recíproco.

Ahora consideremos los polinomios $g_1(x) = x + \omega^2$, y $g_2(x) = x + \omega$. Note que ambos polinomios no son auto-recíprocos, luego,

$$g_1(x)g_2(x) = x^2 + x + 1$$

y

$$g_1^*(x)g_2^*(x) = (1 + x\omega^2)(1 + x\omega) = 1 + x + x^2.$$

Por lo tanto, $g_1(x)g_2(x)$ es auto-recíproco.

□

Teorema 3.5.1. *Sea $C = \langle r(x)\rangle$ un código cíclico aditivo sobre $GF(4)$. El código C es reversible si y solo si $r(x)$ es auto-recíproco.*

Demostración. La prueba de este teorema es similar al teorema 3.4.4. □

Teorema 3.5.2. *Sea $C = \langle \omega p(x) + q(x)\rangle$ un código cíclico aditivo de longitud n sobre $GF(4)$ con $grad(p(x)) = r$. Entonces C es reversible si y solo si $p(x)$ es un polinomio binario auto-recíproco y $C = \langle \omega p(x)\rangle$ ó $C = \langle \overline{\omega} p(x)\rangle$.*

Demostración. Sea $C = \langle \omega p(x) + q(x)\rangle$ un código cíclico aditivo de longitud n sobre $GF(4)$. Supongamos que C es reversible. Consideremos nuevamente la función

$$T : C \to \mathbb{Z}_2^{(n)}[x],$$

definida como en la prueba del Teorema 3.4.3. La imagen de C es un código binario cíclico generado por $p(x)$, veamos que como C es reversible, entonces $T(C)$ también es reversible.
Sea $a(x) \in T(C)$, entonces existe $A(x) \in C$ tal que $a(x) = T(A(x))$, como C es reversible, $A^*(x) \in C$, así,

$$a^*(x) = [T(A(x))]^* = T(A^*(x))$$

Por lo tanto, $a^*(x) \in T(C)$, es decir, $T(C)$ es reversible, y por el Teorema 3.4.4, $p(x)$ es auto-recíproco.

Usando la notación del teorema principal sobre códigos aditivos, el Teorema 3.4.3, el código $C = \langle \omega p(x) + q(x), r(x) \rangle$. En este caso se tiene que $r(x) = 0$, y sabemos por el mismo teorema que $\frac{q(x)(x^n-1)}{p(x)} = r(x)k(x)$, para algún $k(x)$ en $\mathbb{Z}_2^{(n)}[x]$ entonces $\frac{q(x)(x^n-1)}{p(x)} = 0$. Esto implica que $\frac{q(x)(x^n-1)}{p(x)} \in \langle x^n - 1 \rangle$, por tanto $\frac{q(x)(x^n-1)}{p(x)} = (x^n-1)d(x)$, de aquí que $q(x) = p(x)d(x)$, para algún polinomio binario $d(x)$. Como C es reversible,

$$[\omega p(x) + q(x)]^* = [\omega p(x) + p(x)d(x)]^* = \omega x^{grad(d(x))} p(x) + p(x)d^*(x) \in C,$$

esto implica que,

$$\omega x^{grad(d(x))} p(x) + p(x)d^*(x) = [\omega p(x) + q(x)]f(x),$$

entonces la única forma de que esto ocurra es si $f(x) = 1$, es decir,

$$\omega x^{grad(d(x))} p(x) + p(x)d^*(x) = [\omega p(x) + p(x)d(x)],$$

y para que esto pase debe ocurrir que $grad(d(x)) = 0$ y como es binario, $d(x) = 0$, ó $d(x) = 1$. Por lo tanto, si $d(x) = 0$, $C = \langle \omega p(x) \rangle$, y si $d(x) = 1$, tenemos que $C = \langle (\omega + 1)p(x) \rangle = \langle \overline{\omega} p(x) \rangle$.

Recíprocamente, supongamos que $p(x)$ es un polinomio binario auto-recíproco de grado r y $C = \langle \omega p(x) \rangle$ ó $C = \langle \overline{\omega} p(x) \rangle$. Como C no tiene polinomios binarios, entonces $ker(T) = 0$. Luego, T es una función inyectiva. Así, $T(C) = \langle p(x) \rangle$ es un código binario cíclico con base, $\beta = \{p(x), xp(x), \ldots, x^{n-r-1}p(x)\}$. Esto implica que $T^{-1}(\beta) = \{\omega p(x), x\omega p(x), \ldots, x^{n-r-1}\omega p(x)\}$ es una base para C.

Ahora mostraremos que para todo $c(x) \in C$, $c^*(x) \in C$; sea $c(x) \in C$, entonces $c(x) = f(x)\omega p(x)$, donde $grad(f(x)) \leq n-r-1$. Podremos asumir que $grad(c(x)) = n-i$ y $grad(f(x)) = n-r-i$, para algún entero i que satisface $0 < i < n-r+1$. Entonces,

$$\begin{aligned} c^*(x) &= x^{n-i} c(x^{-1}) \\ &= x^{n-i} f(x^{-1}) \omega p(x^{-1}) \\ &= x^{n-r-i} f(x^{-1}) \omega x^r p(x^{-1}) \\ &= x^{n-r-i} f(x^{-1}) \omega p^*(x) \\ &= f^*(x) \omega p(x) \in C. \end{aligned}$$

Por lo tanto, C es reversible. □

Para la demostración del siguiente resultado es necesario darse cuenta de algo muy particular que ocurre en el campo $GF(4)$:

Tomemos $c(x) = c_0 + c_1x + \cdots + c_{n-1}x^{n-1}$ una palabra código en C, y su complemento $c^c(x) = {c_0}^c + {c_1}^c x + \cdots + {c_{n-1}}^c x^{n-1}$. Notemos que para todo $a \in GF(4)$, se tiene que $a + a^c = 1$. Esto implica que,

$$c(x) + c^c(x) = 1 + x + x^2 + \cdots + x^{n-1}.$$

Lema 3.5.2. *Un código lineal cíclico $C = \langle g(x) \rangle$ de longitud n, con n impar, sobre $GF(4)$ es complemento si y solo si $(x-1)$ no divide a $g(x)$.*

Demostración. Sea $C = \langle g(x) \rangle$ un código lineal cíclico sobre $GF(4)$. Como el código C es lineal, entonces C es complemento si y sólo si $c^c(x) \in C$ para todo $c(x) \in C$ si y solo si $c(x) + c^c(x) \in C$, esto ocurre si y solo si, $\frac{x^n-1}{x-1} \in C$, (puesto que $x^n - 1 = (x-1)(c(x) + c^c(x))$) si y solo si $\frac{x^n-1}{x-1} = g(x)f(x)$, para algún polinomio $f(x)$, si y solo si $x^n - 1 = (x-1)g(x)f(x)$. Como n es impar, $x^n - 1$ es producto de polinomios irreducibles distintos. Por lo tanto, $c^c(x) \in C$ si y solo si $x - 1$ no divide a $g(x)$. □

Lema 3.5.3. *Sea $C = \langle \omega p(x) + q(x) \rangle$, un código cíclico aditivo de longitud n sobre $GF(4)$ con $grad(p(x)) = r$. Entonces, C no puede ser complemento.*

Demostración. Sea C un código cíclico aditivo de longitud n sobre $GF(4)$, generado por $\omega p(x) + q(x)$, por ello, C no tiene polinomios binarios. Tomemos $c(x) \in C$, como $c(x) + c^c(x) = 1 + x + x^2 + \cdots + x^{n-1}$ es un polinomio binario, entonces no está en C. Por ser C aditivo, C no puede ser complemento. □

Lema 3.5.4. *Sean C, A y B códigos aditivos tales que $C = A + B$. Si A y B son reversibles, entonces C es reversible.*

Demostración. Supongamos que A y B son reversibles. Tomemos $c(x) \in C$, luego existen $a(x) \in A, b(x) \in B$ tales que $c(x) = a(x) + b(x)$.

Así, por el Lema 3.4.1, $c^*(x) = a^*(x) + x^{grad(a(x))-grad(b(x))}b^*(x) \in C$ pues $a^*(x) \in A$, y $x^{grad(a(x))-grad(b(x))}b^*(x) \in B$ por ser A y B reversibles.

Por lo tanto, C es reversible. □

El recíproco del Lema 3.5.4 no es cierto.

Ejemplo. *Sean $p(x) = x^4 + x^3 + x^2 + x + 1$, $q(x) = x^3 + x$, $r(x) = x^4 + x^3 + x^2 + x + 1$. Entonces $C = \langle \omega p(x) + q(x), r(x) \rangle$ es reversible pero $A = \langle \omega p(x) + q(x) \rangle$ no es reversible.*

Teorema 3.5.3. *Sea $C = \langle \omega p(x), r(x) \rangle$, o $C = \langle \overline{\omega} p(x), r(x) \rangle$, un código cíclico aditivo de longitud n sobre $GF(4)$. El código C es complemento reversible si y solo si $p(x)$, $r(x)$ son auto-recíprocos y $r(x)$ no es múltiplo de $x - 1$.*

Demostración. Supongamos primero que $p(x)$ y $r(x)$ son auto-recíprocos y que $r(x)$ no es múltiplo de $x-1$, entonces, por el Teorema 3.5.1 y el Teorema 3.5.2, se tiene que $A = \langle \omega p(x) \rangle$, $B = \langle r(x) \rangle$ son ambos reversibles. Por el Lema 3.5.4, C es reversible. Además, como $r(x)$ no es múltiplo de $x-1$, entonces por el Lema 3.5.2, B es complemento, entonces $1 + x + \cdots + x^{n-1} \in B$, y por tanto también está en C. Para ver que C es complemento, tomemos $c(x) \in C$, entonces

$$c(x) + c^c(x) = 1 + x + x^2 + \cdots + x^{n-1} \in C.$$

Como C es aditivo $c^c(x) \in C$, es decir, C es complemento.

Recíprocamente, supongamos que C es complemento reversible, consideremos nuevamente la función

$$T : C \to \mathbb{Z}_2^{(n)}[x],$$

definida como en la prueba del Teorema 3.4.3. La imagen de C es un código binario cíclico generado por $p(x)$, y como ya vimos en la demostración del Teorema 3.5.2 si C es reversible, entonces $T(C)$ también es reversible, y por el Teorema 3.4.4 $p(x)$ es auto-recíproco.

Además probemos que si C es reversible, entonces $ker(T) = \langle r(x) \rangle$ también lo es, para ello tomemos $t(x) \in C$ tal que $t(x) \in ker(T)$, como C es reversible, $t^*(x) \in C$, y como además es un polinomio binario, $t^*(x) \in ker(T)$, por lo tanto $ker(T)$ es un código reversible y por el Teorema 3.4.4 $r(x)$ es auto-recíproco.

Del hecho de que C es complemento, por el Lema 3.5.2 se tiene que $r(x)$ no es múltiplo de $x-1$. □

Teorema 3.5.4. *Sea $C = \langle \omega p(x) + q(x), r(x) \rangle$ un código cíclico aditivo de longitud n sobre $GF(4)$ con $q(x) \neq 0$. C es un código cíclico aditivo complemento reversible si y solo si $p(x)$ es auto-recíproco, $r(x)$ es auto-recíproco y no es múltiplo de $(x-1)$ y*

1. *Si $grad(q(x)) = grad(p(x))$, entonces $q(x)$ es auto-recíproco.*
2. *Si $grad(q(x)) < grad(p(x))$, entonces $r(x)$ divide a*

$$[x^{grad(p(x))-grad(q(x))} q^*(x) + q(x)].$$

3. *Si el $grad(q(x)) > grad(p(x))$ entonces $r(x)$ divide a*

$$[x^{grad(q(x))-grad(p(x))} q(x) + q^*(x)].$$

Demostración. Sea $C = \langle \omega p(x) + q(x), r(x) \rangle$ con $q(x) \neq 0$ un código cíclico aditivo de longitud n sobre $GF(4)$.

Suppngamos que C es un código aditivo complemento cíclico reversible sobre $GF(4)$. Aplicando el teorema 3.5.3, $p(x)$ y $r(x)$ son auto-recíprocos, y $r(x)$ no es múltiplo de $(x-1)$.

1. Si $grad(p(x)) = grad(q(x))$, entonces

$$\begin{aligned}[\omega p(x) + q(x)]^* &= \omega p^*(x) + x^{grad(p(x)-grad(q(x)))} q^*(x) \\ &= \omega p(x) + q^*(x).\end{aligned}$$

   Por lo tanto $q(x) = q^*(x)$.

2. Si $grad(q(x)) < grad(p(x))$, entonces

$$\begin{aligned}[\omega p(x) + q(x)]^* &= \omega p^*(x) + x^{grad(p(x)-grad(q(x)))} q^*(x) \\ &= \omega p(x) + x^{grad(p(x))-grad(q(x))} q^*(x) \in C.\end{aligned}$$

   Esto por ser C reversible, luego,

$$\omega p(x) + x^{grad(p(x))-grad(q(x))} q^*(x) = [\omega p(x) + q(x)] f_1(x) + r(x) f_2(x),$$

   para algunos polinomios $f_1(x), f_2(x)$. Como $p(x), r(x), q(x)$ son binarios, y por la igualdad anterior $f_1(x)$ debe ser tambien un polinomio binario. Veamos los grados de estas sumas de polinomios,

$$grad(\omega p(x) + q(x))^* \leq grad(\omega p(x) + q(x)) + grad(f_1(x)) \leq grad(\omega p(x) + q(x)),$$

   entonces $grad(f_1(x)) \leq 0$, por tanto es un polinomio constante, como es binario no queda más remedio que $f_1(x) = 1$.

   Con todo esto, $\omega p(x) + x^{grad(p(x))-grad(q(x))} q^*(x) = \omega p(x) + q(x) + r(x) f_2(x)$, es decir, $q(x) + x^{grad(p(x))-grad(q(x))} q^*(x) = r(x) f_2(x)$.

3. Esta prueba es similar a la anterior.

Recíprocamente, supongamos que $p(x)$ y $r(x)$ son auto-recíprocos, y $r(x)$ no es múltiplo de $(x-1)$.

1. Supongamos que $grad(p(x)) = grad(q(x))$ y $p(x), q(x), r(x)$ son auto-recíprocos, tomemos $c(x) \in C$, entonces

$$c(x) = [\omega p(x) + q(x)] f_1(x) + r(x) f_2(x),$$

   para algunos polinomios $f_1(x), f_2(x)$. Sin pérdida de generalidad pongamos $i = grad((\omega p(x) + q(x))(f_1(x))) - grad(r(x) f_2(x))$, así,

$$\begin{aligned}c^*(x) &= [\omega p(x) + q(x)]^* f_1(x)^* + x^i f_2(x)^* r^*(x) \\ &= [\omega p^*(x) + q^*(x)] f_1^*(x) + r(x) x^i f_2^*(x) \\ &= [\omega p(x) + q(x)] f_1^*(x) + r(x)[x^i f_2^*(x)] \in C.\end{aligned}$$

   Por lo tanto C es reversible. Como $r(x)$ no es un múltiplo de $(x-1)$, por el Teorema 3.5.3 C es complemento.

2. Supongamos que $grad(q(x)) < grad(p(x))$, $p(x)$ y $r(x)$ son auto-recíprocos, $r(x)$ divide a $x^{grad(p(x))-grad(q(x))}q^*(x)+q(x)$, es decir, existe un polinomio $f_3(x)$ tal que $x^{grad(p(x))-grad(q(x))}q^*(x)+q(x) = r(x)f_3(x)$, y además $r(x)$ no es múltiplo de $(x-1)$.

   Sea $c(x) \in C$, entonces $c(x) = [\omega p(x) + q(x)]f_1(x) + r(x)f_2(x)$, para algunos polinomios $f_1(x), f_2(x)$. Nuevamente sin pérdida de generalidad pongamos $i = grad((\omega p(x)+q(x))(f_1(x)))-grad(r(x)f_2(x))$, entonces,

$$\begin{aligned}
c^*(x) &= [\omega p(x) + q(x)]^* f_1(x)^* + x^i f_2(x)^* r^*(x) \\
&= [\omega p^*(x) + q^*(x)]f_1^*(x) + r(x)x^i f_2^*(x) \\
&= [\omega p(x) + x^{grad(p(x))-grad(q(x))}q^*(x)]f_1^*(x) + r(x)[x^i f_2^*(x)] \\
&= \omega p(x)f_1^*(x) + q(x)f_1^*(x) + q(x)f_1^*(x) \\
&+ x^{grad(p(x))-grad(q(x))}q^*(x)f_1^*(x) + x^i r(x)f_2^*(x) \\
&= [\omega p(x) + q(x)]f_1^*(x) + [q(x) + x^{grad(p(x))-grad(q(x))}q^*(x)]f_1^*(x) \\
&+ x^i r(x)f_2^*(x) \\
&= [\omega p(x) + q(x)]f_1^*(x) + r(x)f_3(x)f_1^*(x) + x^i r(x)f_2^*(x) \\
&= [\omega p(x) + q(x)]f_1^*(x) + r(x)[f_3(x)f_1^*(x) + x^i f_2^*(x)] \in C.
\end{aligned}$$

   Para algún polinomio $f_3(x)$. Por lo tanto $c^*(x) \in C$ y esto implica que C es reversible. Como $r(x)$ no divide a $(x-1)$, por el Teorema 3.5.3, C es complemento.

3. La prueba de este inciso es similar al anterior.

□

3.6. Códigos aditivos de longitud 7

En esta sección usaremos la teoría desarrollada a lo largo de este capítulo para estudiar códigos aditivos complementos cíclicos reversibles sobre $GF(4)$. Presentaremos algunos ejemplos de códigos de longitud 7.

Encontremos primero los divisores de $x^7 - 1$, es decir, como

$$x^7 - 1 = (x+1)(x^3 + x^2 + 1)(x^3 + x + 1),$$

donde $x+1,\ x^3+x^2+1,\ x^3+x+1$ son polinomios irreducibles; entonces los divisores de x^7-1

son,

$$p_0(x) = 1$$
$$p_1(x) = x + 1$$
$$p_2(x) = x^3 + x^2 + 1$$
$$p_3(x) = x^3 + x + 1$$
$$p_4(x) = x^4 + x^3 + x^2 + 1 = (x + 1)(x^3 + x + 1)$$
$$p_5(x) = x^4 + x^2 + x + 1 = (x + 1)(x^3 + x^2 + 1)$$
$$p_6(x) = x^6 + x^5 + x^4 + x^3 + x^2 + x + 1 = (x^3 + x^2 + 1)(x^3 + x + 1).$$

Ahora calculemos el recíproco de estos polinomios,

$$p_0^*(x) = 1$$
$$p_1^*(x) = x + 1$$
$$p_2^*(x) = x^3 + x + 1$$
$$p_3^*(x) = x^3 + x^2 + 1$$
$$p_4^*(x) = x^4 + x^2 + x + 1$$
$$p_5^*(x) = x^4 + x^3 + x^2 + 1$$
$$p_6^*(x) = x^6 + x^5 + x^4 + x^3 + x^2 + x + 1.$$

De aquí que, los polinomios p_0, p_1, p_6 son auto-recíprocos.

Ahora veamos los códigos cíclicos lineales generados por los polinomios divisores de $x^7 - 1$. Aplicando el teorema 3.4.1 se pueden generar 7 códigos cíclicos lineales. En la tabla 3.1 podemos ver dichos códigos, su dimensión y su número de elementos. Al grado del polinomio generador lo denotaremos como r.

Código	Dimensión($k = n - r$)	Tamaño($M = 4^k$)
$C_0(x) = \langle 1 \rangle$	$k = 7 - 0 = 7$	$M = 4^7 = 16,384$
$C_1(x) = \langle x + 1 \rangle$	$k = 7 - 1 = 6$	$M = 4^6 = 4,096$
$C_2(x) = \langle x^3 + x^2 + 1 \rangle$	$k = 7 - 3 = 4$	$M = 4^4 = 256$
$C_3(x) = \langle x^3 + x + 1 \rangle$	$k = 7 - 3 = 4$	$M = 256$
$C_4(x) = \langle x^4 + x^3 + x^2 + 1 \rangle$	$k = 7 - 4 = 3$	$M = 4^3 = 64$
$C_5(x) = \langle x^4 + x^2 + x + 1 \rangle$	$k = 7 - 4 = 3$	$M = 64$
$C_6(x) = \langle x^6 + x^5 + x^4 + x^3 + x^2 + x + 1 \rangle$	$k = 7 - 6 = 1$	$M = 4$

Tabla 3.1: Códigos cíclicos de longitud 7.

Como estos son códigos lineales, son subespacios vectoriales de $GF(4)^7$, por lo cual, es posible dar una base para ellos, y aquí enlistamos dichas bases:

- Para C_0, en $GF(4)^{(7)}[x]$ la base es $\{1, x, x^2, x^3, x^4, x^5, x^6\}$ y su correspondiente base en $GF(4)^7$ es

$$\{1000000, 0100000, 0010000, 0001000, 0000100, 0000010, 0000001\}.$$

  Para este código la distancia es $d(C_0) = 1$.

- Para C_1, en $GF(4)[x]^{(7)}$ la base es $\{x+1, x^2+x, x^3+x^2, x^4+x^3, x^5+x^4, x^6+x^5\}$, y su base correspondiente en $GF(4)^7$ es

$$\{1100000, 0110000, 0011000, 0001100, 0000110, 0000011\}.$$

  La distancia del código es $d(C_1) = 2$.

- Para C_2, en $GF(4)[x]^{(7)}$ la base es $\{x^3+x^2+1, x^4+x^3+x, x^5+x^4+x^2, x^6+x^5+x^3\}$ y en $GF(4)^7$, la base correspondiente es

$$\{1011000, 0101100, 0010110, 0001011\}.$$

  Para C_2 la distancia del código es $d(C_2) = 3$.

- Para C_3, en $GF(4)[x]^{(7)}$ la base es $\{x^3+x+1, x^4+x^2+x, x^5+x^3+x^2, x^6+x^4+x^3\}$ y en $GF(4)^7$, la base correspondiente es

$$\{1101000, 0110100, 0011010, 0001101\}.$$

  La distancia de C_3 es $d(C_3) = 3$.

- Para C_4, la base en $GF(4)[x]^{(7)}$ es $\{x^4+x^3+x^2+1, x^5+x^4+x^3+x, x^6+x^5+x^4+x^2\}$, la correspondiente base en $GF(4)^7$ es

$$\{1011100, 0101110, 0010111\}.$$

  La distancia de C_4 es $d(C_4) = 4$.

- Para C_5, la base en $GF(4)[x]^{(7)}$ es $\{x^4+x^2+x+1, x^5+x^4+x^2+x, x^6+x^4+x^3+x^2\}$, la correspondiente base en $GF(4)^7$ es

$$\{1110100, 0111010, 0011101\}.$$

  Para el código C_5, la distancia es $d(C_5) = 4$.

- Para C_6, la base en $GF(4)[x]^{(7)}$ es $\{x^6+x^5+x^4+x^3+x^2+x+1\}$, y en $GF(4)^7$ le corresponde la base $\{1111111\}$. Este código tiene distancia $d(C_6) = 7$.

Ahora encontraremos los códigos aditivos cíclicos aplicando el teorema 3.4.3, aquí tenemos que $r(x)$ serán también los polinomios $p_i(x)$, con $i \in \{0,1,2,3,4,5,6\}$, pues debe ocurrir que $r(x)$ también sea divisor de x^7-1.Aquí enunciamos algunos de ellos para $p_0(x)=1$.

- $r(x)=1$, notemos que el $grad(r(x))=0$, y debe ocurrir que $grad(q(x)) < grad(r(x))=0$, lo cual no es posible, por tanto, $r(x)\neq 1$.

- $r(x)=x+1$. Como $grad(q(x))<1$, entonces se tiene que $q(x)=0,\ q(x)=1$, es decir,

Código	Dimensión(k)	Tamaño($M=2^k$)	Distancia
$C_1(x)=\langle \omega+0, x+1\rangle$	$k=2(7)-0-1=13$	$M=2^{13}=8,192$	$d(C_1)=1$
$C_2(x)=\langle \omega+1, x+1\rangle$	$k=2(7)-0-1=13$	$M=2^{13}=8,192$	$d(c_2)=1$

Tabla 3.2: Códigos cíclicos aditivos de longitud 7 y tamaño $8,192$.

- $r(x)=x^3+x^2+1$. Como $grad(q(x))<3$, entonces se tienen los casos:

  - $grad(q(x))=0$, ocurre que $q(x)=0$ ó $q(x)=1$.
  - $grad(q(x))=1$, ocurre que $q(x)=x$ ó $q(x)=x+1$.
  - $grad(q(x))=2$, ocurre que $q(x)=x^2$, $q(x)=x^2+x+1$, $q(x)=x^2+1$, ó $q(x)=x^2+x$.

- $r(x)=x^3+x+1$. Como $grad(q(x))<3$, entonces se tienen los casos:

  - $grad(q(x))=0$, ocurre que $q(x)=0$ ó $q(x)=1$.
  - $grad(q(x))=1$, ocurre que $q(x)=x$, ó $q(x)=x+1$.
  - $grad(q(x))=2$, ocurre que $q(x)=x^2$, $q(x)=x^2+x+1$, $q(x)=x^2+1$, ó $q(x)=x^2+x$.

Código	Dimensión	Tamaño ($M = 2^k$)	Distancia
$C_3(x) = \langle \omega + 0, x^3 + x^2 + 1 \rangle$	$k = 2(7) - 0 - 3 = 11$	$M = 2^{11} = 2,048$	$d(C_3) = 1$
$C_4(x) = \langle \omega + 1, x^3 + x^2 + 1 \rangle$	$k = 2(7) - 0 - 3 = 11$	$M = 2^{11} = 2,048$	$d(C_4) = 1$
$C_5(x) = \langle \omega + x, x^3 + x^2 + 1 \rangle$	$k = 2(7) - 0 - 3 = 11$	$M = 2^{11} = 2,048$	$d(C_5) = 2$
$C_6(x) = \langle \omega + (x+1), x^3 + x^2 + 1 \rangle$	$k = 2(7) - 0 - 3 = 11$	$M = 2^{11} = 2,048$	$d(C_6) = 2$
$C_7(x) = \langle \omega + (x^2), x^3 + x^2 + 1 \rangle$	$k = 2(7) - 0 - 3 = 11$	$M = 2^{11} = 2,048$	$d(C_7) = 2$
$C_8(x) = \langle \omega + (x^2 + x + 1), x^3 + x^2 + 1 \rangle$	$k = 2(7) - 0 - 3 = 11$	$M = 2^{11} = 2,048$	$d(C_8) = 2$
$C_9(x) = \langle \omega + (x^2 + 1), x^3 + x^2 + 1 \rangle$	$k = 2(7) - 0 - 3 = 11$	$M = 2^{11} = 2,048$	$d(C_9) = 2$
$C_{10}(x) = \langle \omega + (x^2 + x), x^3 + x^2 + 1 \rangle$	$k = 2(7) - 0 - 3 = 11$	$M = 2^{11} = 2,048$	$d(C_{10}) = 2$

Tabla 3.3: Códigos cíclicos aditivos de longitud 7 y tamaño $2,048$.

Código	Dimensión	Tamaño ($M = 2^k$)	Distancia
$C_{11}(x) = \langle \omega + 0, x^3 + x + 1 \rangle$	$k = 2(7) - 0 - 3 = 11$	$M = 2^{11} = 2,048$	$d(C_{11}) = 1$
$C_{12}(x) = \langle \omega + 1, x^3 + x + 1 \rangle$	$k = 2(7) - 0 - 3 = 11$	$M = 2^{11} = 2,048$	$d(C_{12}) = 1$
$C_{13}(x) = \langle \omega + x, x^3 + x + 1 \rangle$	$k = 2(7) - 0 - 3 = 11$	$M = 2^{11} = 2,048$	$d(C_{13}) = 2$
$C_{14}(x) = \langle \omega + (x+1), x^3 + x + 1 \rangle$	$k = 2(7) - 0 - 3 = 11$	$M = 2^{11} = 2,048$	$d(C_{14}) = 2$
$C_{15}(x) = \langle \omega + (x^2), x^3 + x + 1 \rangle$	$k = 2(7) - 0 - 3 = 11$	$M = 2^{11} = 2,048$	$d(C_{15}) = 2$
$C_{16}(x) = \langle \omega + (x^2 + x + 1), x^3 + x + 1 \rangle$	$k = 2(7) - 0 - 3 = 11$	$M = 2^{11} = 2,048$	$d(C_{16}) = 2$
$C_{17}(x) = \langle \omega + (x^2 + 1), x^3 + x + 1 \rangle$	$k = 2(7) - 0 - 3 = 11$	$M = 2^{11} = 2,048$	$d(C_{17}) = 2$
$C_{18}(x) = \langle \omega + (x^2 + x), x^3 + x + 1 \rangle$	$k = 2(7) - 0 - 3 = 11$	$M = 2^{11} = 2,048$	$d(C_{18}) = 2$

Tabla 3.4: Códigos cíclicos aditivos de longitud 7 y tamaño $2,048$.

Hasta aquí hemos encontrado 18 códigos, falta hacer el cálculo para cuando $r(x) = x^4 + x^3 + x^2 + 1$, $r(x) = x^4 + x^3 + x + 1$, ó $r(x) = x^6 + x^5 + x^4 + x^3 + x^2 + x + 1$, de ahí surgen los códigos faltantes correspondientes a $p_0(x) = 1$.

Ahora de la tabla 3.1, elegiremos aquellos que son generados por polinomios auto-recíprocos y los ponemos en la tabla 3.5.

Código	Dimensión(k)	Tamaño(M)	Distancia
$C_0(x) = \langle 1 \rangle$	$k = 7 - 0 = 7$	$M = 4^7 = 16,384$	$d(C_0) = 1$
$C_1(x) = \langle x + 1 \rangle$	$k = 7 - 1 = 6$	$M = 4^6 = 4,096$	$d(C_1) = 2$
$C_6(x) = \langle x^6 + x^5 + x^4 + x^3 + x^2 + x + 1 \rangle$	$k = 7 - 6 = 1$	$M = 4$	$d(C_6) = 7$

Tabla 3.5: Códigos lineales cíclicos reversibles de longitud 7.

Por el teorema 3.4.4 $C_0,\ C_1,\ C_6$ son reversibles.

Consideremos ahora los códigos aditivos de la forma $C = \langle \omega p(x) + q(x) \rangle$, en este caso como $r(x) = 0$, entonces $q(x)$ puede ser cualquier polinomio binario de grado 6, por el teorema 3.5.2, para que este tipo de códigos sean reversibles debe ocurrir que $p(x)$ sea auto-recíproco y $q(x) = 0$, además como $r(x) = 0$, la dimensión del código es $k = 7 - grad(p(x))$. Por lo tanto, en la tabla 3.6 tenemos los polinomios aditivos cíclicos reversibles que no tienen polinomios binarios.

Código	Dimensión(k)	Tamaño(M)	Distancia
$C_0(x) = \langle \omega \rangle$	$k = 7-0 = 7$	$M = 2^7 = 128$	$d(C_0) = 1$
$C_1(x) = \langle \omega(x+1) \rangle$	$k = 7-1 = 6$	$M = 2^6 = 64$	$d(C_1) = 2$
$C_6(x) = \langle \omega(x^6+x^5+x^4+x^3+x^2+x+1) \rangle$	$k = 7-6 = 1$	$M = 2^1 = 2$	$d(C_6) = 7$
$C_0'(x) = \langle \overline{\omega} \rangle$	$k = 7-0 = 7$	$M = 2^7 = 128$	$d(C_0') = 1$
$C_1'(x) = \langle \overline{\omega}(x+1) \rangle$	$k = 7-1 = 6$	$M = 2^6 = 64$	$d(C_1') = 2$
$C_6'(x) = \langle \overline{\omega}(x^6+x^5+x^4+x^3+x^2+x+1) \rangle$	$k = 7-6 = 1$	$M = 2^1 = 2$	$d(C_6') = 7$

Tabla 3.6: Códigos aditivos cíclicos reversibles de longitud 7.

Enseguida utilizaremos el lema 3.5.2 para elegir códigos de longitud 7, lineales complemento cíclicos. De la tabla 3.1 elegimos aquellos códigos generados por un polinomio que no sea múltiplo de $x - 1$, es decir,

Código	Dimensión(k)	Tamaño(M)	Distancia
$C_2(x) = \langle x^3 + x^2 + 1 \rangle$	$k = 7-3 = 4$	$M = 4^4 = 256$	$d(C_2) = 3$
$C_3(x) = \langle x^3 + x + 1 \rangle$	$k = 7-3 = 4$	$M = 256$	$d(C_3) = 3$
$C_6(x) = \langle x^6 + x^5 + x^4 + x^3 + x^2 + x + 1 \rangle$	$k = 7-6 = 1$	$M = 4$	$d(C_6) = 7$

Tabla 3.7: Códigos cíclicos complemento de longitud 7.

Para el caso de códigos lineales, de longitud $n = 7$, sólo hay un código lineal complemento cíclico reversible, y es generado por el polinomio $p(x) = x^6 + x^5 + x^4 + x^3 + x^2 + x + 1$ con dimensión $k = 7 - 6 = 1$ y distancia $d(C_6) = 7$, por lo tanto, el tamaño de este código es $M = 4^1 = 4$, que es el número de palabras código.

Ahora aplicaremos el teorema 3.5.3 a los códigos de las tablas 3.2, 3.3, 3.4 para elegir de ellos cuáles son complemento, es decir, elegimos aquellos donde $p_i(x), r(x)$ son auto-recíprocos, y además $q(x) = 0$. Notemos que la única opción para la elección de $r(x)$, para que sea auto-recíproco y que no sea múltiplo de $x - 1$, es $r(x) = x^6 + x^5 + x^4 + x^3 + x^2 + x + 1$.

Código	Dimensión(k)	Tamaño(M)	Distancia
$C_1(x) = \langle \omega + 0, x^6 + x^5 + x^4 + x^3 + x^2 + x + 1 \rangle$	$k = 2(7) - 0 - 6 = 8$	$M = 2^8 = 256$	$d(C_1) = 1$
$C_2(x) = \langle \omega(x+1) + 0, x^6 + x^5 + x^4 + x^3 + x^2 + x + 1 \rangle$	$k = 2(7) - 1 - 6 = 7$	$M = 2^7 = 128$	$d(C_2) = 2$
$C_6(x) = \langle \omega(x^6 + x^5 + x^4 + x^3 + x^2 + x + 1) + 0, x^6 + x^5 + x^4 + x^3 + x^2 + x + 1 \rangle$	$k = 2(7) - 6 - 6 = 2$	$M = 2^2 = 4$	$d(C_6) = 7$

Tabla 3.8: Códigos aditivos complemento cíclico reversible de longitud 7.

Finalmente veamos que por el teorema 3.5.4, es necesario que $p(x)$ y $r(x)$ sean auto-recíprocos y $r(x)$ no sea múltiplo de $x - 1$. Así, $p(x)$ podría ser $1, x + 1, x^6 + x^5 + x^4 + x^3 + x^2 + x + 1$, y $r(x)$ sólo puede ser $x^6 + x^5 + x^4 + x^3 + x^2 + x + 1$.

Sin embargo, aún nos faltan condiciones para el polinomio $q(x)$, las cuales están dadas también por el Teorema 3.5.4.

Si $p(x) = 1$ y $r(x) = x^6 + x^5 + x^4 + x^3 + x^2 + x + 1$.

1. Si el $grad(q(x)) = 0$, entonces $q(x) = 1$, el cual es auto-recíproco. Por lo tanto, el código aditivo complemento cíclico reversible generado es:

$$C_1 = \langle \omega + 1, x^6 + x^5 + x^4 + x^3 + x^2 + x + 1 \rangle,$$

   para este código los elementos generadores en $GF(4)^7$, correspondientes a los polinomios generadores $p(x) = 1$, $r(x) = x^6+x^5+x^4+x^3+x^2+x+1$, $q(x) = 0$, son $1000000, 1111111, 0000000$. La distancia de C_1 es $d(C_1) = 1$.

2. $grad(q(x)) < grad(p(x)) = 0$, esto no puede ser.

3. Si $0 = grad(p(x)) < grad(q(x))$, se tienen distintos casos:

   - $grad(q(x)) = 1$, no hay polinomios $q(x)$ que cumplan las condiciones del teorema 3.5.4.

- $grad(q(x)) = 2$, no hay polinomios $q(x)$ que cumplan las condiciones del teorema 3.5.4.
- $grad(q(x)) = 3$, no hay polinomios $q(x)$ que cumplan las condiciones del teorema 3.5.4.
- $grad(q(x)) = 4$, los polinomios $q(x)$ que cumplen las condiciones del teorema 3.5.4 son los siguientes: $x^4 + x^3$, $x^4 + x^3 + 1$, es decir, como $x^4(q(x)) + q^*(x) = 0$, entonces $r(x)$ divide a $x^4(q(x)) + q^*(x) = 0$. Por lo tanto, los códigos aditivos complemento cíclico reversibles son

  $$C_2 = \langle \omega + (x^4 + x^3), x^6 + x^5 + x^4 + x^3 + x^2 + x + 1 \rangle,$$
  $$C_3 = \langle \omega + (x^4 + x^3 + 1), x^6 + x^5 + x^4 + x^3 + x^2 + x + 1 \rangle.$$

  La distancia de C_2 es $d(C_2) = 3$ y la distancia de C_3 es $d(C_3) = 3$. Para estos códigos los elementos generadores en $GF(4)^7$ correspondientes a los polinomios $p(x)$, $r(x)$, $q(x)$, son $1000000, 1111111, 0001100$ y $1000000, 1111111, 1001100$, respectivamente.
- Si $grad(q(x)) = 5$, los polinomios $q(x)$ que cumplen las condiciones del teorema 3.5.4 son los siguientes: $x^5 + x^2$, $x^5 + x^4 + x^3 + x^2$,$x^5 + x^2 + 1$, $x^5 + x^4 + x^3 + x^2 + x + 1$, es decir, $r(x)$ divide a $x^5(q(x)) + q^*(x)$. Por lo tanto, los códigos aditivos complemento cíclico reversibles son

  $$C_4 = \langle \omega + (x^5 + x^2), x^6 + x^5 + x^4 + x^3 + x^2 + x + 1 \rangle,$$
  $$C_5 = \langle \omega + (x^5 + x^4 + x^3 + x^2), x^6 + x^5 + x^4 + x^3 + x^2 + x + 1 \rangle,$$
  $$C_6 = \langle \omega + (x^5 + x^2 + 1), x^6 + x^5 + x^4 + x^3 + x^2 + x + 1 \rangle,$$
  $$C_7 = \langle \omega + (x^5 + x^4 + x^3 + x^2 + 1), x^6 + x^5 + x^4 + x^3 + x^2 + x + 1 \rangle.$$

  Estos códigos tienen distancias $d(C_4) = 3, d(C_5) = 3, d(C_6) = 3, d(C_7) = 3$.

  Los elementos generadores en $GF(4)^7$ para C_4, C_5, C_6, C_7 correspondientes a $p(x)$, $r(x)$, $q(x)$, son $\{1000000, 1111111, 0010010\}$,
  $\{1000000, 1111111, 0011110\}$, $\{1000000, 1111111, 1010010\}$,
  $\{1000000, 1111111, 1111110\}$, respectivamente.

Los códigos generados tienen dimensión $k = 2(7) - 0 - 6 = 8$, y por lo tanto tienen tamaño $M = 2^8 = 256$, que es el número de palabras código.

Los códigos $C_2, C_3, C_4, C_5, C_6, C_7$ tienen distancia 3 y ellos mejoran las codificaciones que ya se tenían para los algoritmos de ADN pues en dichas codificaciones sólo se trabajaban con códigos de tamaño 180 y distancia 3; al tener la misma distancia, es mejor trabajar con códigos que tienen 256 palabras, pues esto evita tener hibridaciones no deseables entre las hebras de ADN.

Ahora si $p(x) = x + 1$ y $r(x) = x^6 + x^5 + x^4 + x^3 + x^2 + x + 1$.

1. Si el $grad(q(x)) = 1$, entonces $q(x) = x$ ó $q(x) = x + 1$, los cuales son auto-recíprocos. Por lo

tanto, los códigos aditivos complemento cíclicos reversibles generados son:

$$C_1 = \langle \omega(x+1) + (x), x^6 + x^5 + x^4 + x^3 + x^2 + x + 1 \rangle,$$
$$C_2 = \langle \omega(x+1) + (x+1), x^6 + x^5 + x^4 + x^3 + x^2 + x + 1 \rangle.$$

Para $C_1,\ C_2$ los elementos generadores en $GF(4)^7$ correspondientes a $p(x)$, $r(x),\ q(x)$, son $\{1100000, 1111111, 0100000\}$ y $\{1100000, 1111111, 1100000\}$ respectivamente. Las distancias de los códigos C_1, C_2 tienen distancia $d(C_1) = 2, d(C_2) = 2$.

2. $grad(q(x)) < grad(p(x)) = 1$, entonces $q(x) = 1$. Luego, $xq^*(x) + q(x) = x + 1$ y $r(x)$ no divide a este polinomio, por lo tanto, el código generado no cumple las condiciones del teorema 3.5.4.

3. Si $1 = grad(p(x)) < grad(q(x))$, se tienen distintos casos:

   - $grad(q(x)) = 2$, no hay polinomios $q(x)$ que cumplan las condiciones del teorema 3.5.4.
   - $grad(q(x)) = 3$, no hay polinomios $q(x)$ que cumplan las condiciones del teorema 3.5.4.
   - $grad(q(x)) = 4$, el polinomios $q(x)$ que cumple las condiciones del teorema 3.5.4 es: x^4, es decir, como $x^4(q(x)) + q^*(x) = 0$, entonces $r(x)$ divide a $x^4(q(x)) + q^*(x) = 0$. Por lo tanto, el códigos aditivo complemento cíclico reversible es

     $$C_3 = \langle \omega(x+1) + (x^4), x^6 + x^5 + x^4 + x^3 + x^2 + x + 1 \rangle.$$

     Para C_3 los elementos generadores en $GF(4)^7$ son $\{1100000, 1111111, 0000100\}$, correspondientes a los polinomios $p(x)$, $r(x)$, $q(x)$ y su distancia es $d(C_3) = 3$.

   - Si $grad(q(x)) = 5$, los polinomios $q(x)$ que cumplen las condiciones del teorema 3.5.4 son los siguientes: $x^5 + x^4 + x^3 + x + 1$, $x^5 + x^3$, $x^5 + x^4 + x^3$, es decir, como $x^4(q(x)) + q^*(x) = 0$, entonces $r(x)$ divide a $x^4(q(x)) + q^*(x) = 0$. Por lo tanto, los códigos aditivos complemento cíclicos reversibles son

     $$C_4 = \langle \omega(x+1) + (x^5 + x^4 + x^3 + x + 1), x^6 + x^5 + x^4 + x^3 + x^2 + x + 1 \rangle,$$
     $$C_5 = \langle \omega(x+1) + (x^5 + x^3), x^6 + x^5 + x^4 + x^3 + x^2 + x + 1 \rangle,$$
     $$C_6 = \langle \omega(x+1) + (x^5 + x^4 + x^3), x^6 + x^5 + x^4 + x^3 + x^2 + x + 1 \rangle.$$

     Los elementos generadores en $GF(4)^7$ para C_4, C_5, C_6 correspondientes a los polinomios $p(x),\ r(x),\ q(x)$, son $\{1100000, 1111111, 1101110\}$, $\{1100000, 1111111, 0001010\}$ y $\{1100000, 1111111, 0001110\}$, respectivamente, y sus distancias son $d(C_4) = 3, d(C_5) = 3, d(C_6) = 3$.

Los códigos generados tienen dimensión $k = 2(7) - 1 - 6 = 7$, y por lo tanto tienen tamaño $M = 2^7 = 128$, que es el número de palabras código.

La última opción para ser el polinomio $p(x)$ para que el código generado sea aditivo cíclico reversible es $p(x) = x^6 + x^5 + x^4 + x^3 + x^2 + x + 1$.

1. Si el $grad(q(x)) = 6$, esto no puede ocurrir puesto que $grad(r(x)) = 6$ y nosotros asumimos que $grad(q(x)) < grad(r(x))$.

2. Si $grad(q(x)) < grad(p(x)) = 6$, se tienen distintos casos:

   - Si $grad(q(x)) = 0$, entonces $q(x) = 1$, pero para este polinomio, $r(x)$, no divide al polinomio $x^6q^*(x) + q(x)$, por lo tanto, el código generado no es un código aditivo complemento cíclico reversible.
   - Si $grad(q(x)) = 1$, entonces $q(x) = x$ y $q(x) = x + 1$, sin embargo estos polinomios tampoco cumplen las condiciones del teorema 3.5.4.
   - $grad(q(x)) = 2$, no hay polinomios $q(x)$ que cumplan las condiciones del teorema 3.5.4.
   - Si $grad(q(x)) = 3$, hay un polinomio $q(x)$ que cumple las condiciones del teorema 3.5.4: $q(x) = x^3$, es decir, como $x^3(q^*(x))+q^(x) = 0$, entonces $r(x)$ divide a $x^3(q^*(x))+q^(x) = 0$, sin embargo $r(x)$ no divide a $\frac{q(x)(x^7-1)}{x^6+x^5+x^4+x^3+x^2+x+1}$, la cual es condición del teorema 3.4.3.
   - Si $grad(q(x)) = 4$, hay un polinomio $q(x)$ que cumple las condiciones del teorema 3.5.4: $q(x) = x^4 + x^2$, es decir, como $x^2(q^*(x)) + q^(x) = 0$, entonces $r(x)$ divide a $x^2(q^*(x)) + q^(x) = 0$, pero como en el inciso anterior, tampoco se cumple la condición del teorema 3.4.3.
   - Si $grad(q(x)) = 5$, los polinomios $q(x)$ que cumplen las condiciones del teorema 3.5.4 son los siguientes: $x^5+x^4+x^2+x$, x^5+x^3+x, x^5+x, $x^5+x^4+x^3+x^2+x$, es decir, como $x(q^*(x)) + q(x) = 0$, entonces $r(x)$ divide a $x(q^*(x)) + q(x) = 0$, sin embargo también se debe cumplir la condición del teorema 3.4.3 acerca de $q(x)$, es decir $r(x)$ debe dividir a $\frac{q(x)(x^7-1)}{x^6+x^5+x^4+x^3+x^2+x+1}$, y para estos polinomios $q(x)$ esto no ocurre.

Por lo tanto, no hay códigos aditivos complemento cíclicos reversibles con $p(x) = r(x) = x^6 + x^5 + x^4 + x^3 + x^2 + x + 1$.

Con todo este análisis queda claro cómo se puede utilizar la teoría desarrollada para construir códigos aditivos complemento cíclicos reversibles, y cuáles deben ser las características de sus polinomios generadores.

Capítulo 4

Aplicaciones de la computación del ADN

A lo largo de este capítulo se describen algunos problemas de teoría de gráficas que han sido resueltos mediante algoritmos de ADN debido a su complejidad. Sin embargo hacemos enfásis en que estos algoritmos sólo se han probado en laboratorios de química con moléculas de ADN reales. Hasta el momento se ha intentado simular estos procesos en una computadora, pero los ejemplos que aquí presentamos fueron resueltos en un laboratorio de química de forma experimental.

Las operaciones del ADN que se utilizarán se describen a continuación, donde s es una hebra simple de ADN y los parámetros T denotan tubos de ensayo, los cuales pueden contener o no moléculas de ADN, cuando no contienen moléculas se dice que son tubos vacíos. En cada operación se indicará si los tubos requeridos son vacíos o no.

- *Extraer*(T, s, T^+, T^-). Los resultados de esta operación estarán en T^+ y en T^-. El tubo T^+ contiene todas las moléculas de T que contienen a s como subsucesión y el resto se encuentran en T^-.

- *Unir*$(T_0, T_1, \ldots, T_n)$. Sean $T_1, \ldots, T_n$ dados, T_0 es la unión de $T_1, \ldots, T_n$.

- *Detectar*(T). Devuelve el valor Verdadero si T contiene al menos una molécula de ADN y Falso de otra forma.

- *Descartar*(T). Si T no es necesitado se desecha.

- *Amplificar*$(T_0, T_1, \ldots, T_n)$. Sean $T_0 \neq \emptyset$ y $T_1, \ldots, T_n$ tubos dados vacíos. Los tubos $T_1, \ldots, T_n$ serán copias idénticas de T_0, al finalizar la operación, $T_0 = \emptyset$.

- *Número*(T). Es el número de diferentes hebras de ADN que hay en T.

- *Seleccionar(T_1, l, T_2).* Dado $T_1 \neq \emptyset$, l un entero positivo y T_2 un tubo vacío. En esta operación todas las hebras de ADN de longitud l que hay en T_1 son removidas y puestas en T_2.

- *Desnaturalizar(T).* Todas las sucesiones de hebras dobles son disociadas en sucesiones de hebras simples de ADN en T.

- *Recocer(T).* Consiste en poner un gran número de nucleótidos y enzima ligasa en T, entonces enfríar el tubo, lo que permite que en T se formen todas las sucesiones factibles de hebras dobles de ADN.

- *Anexar(T,s).* Esta operación se encarga de anexar s al final de cada una de las hebras de ADN que se encuentran en T.

- *Ligación(T_3, T_1, T_2).* Consiste en ligar las hebras del tubo T_1 y T_2 y almacenarlas en T_3.

- *Separación(T, T_v, T_f, θ).* Esta operación consiste en dividir las hebras de T de tal forma que se almacenan en T_v las que resulten verdaderas bajo el criterio θ y en T_f las que resulten falsas.

4.1. El árbol de expansión mínima

En esta sección se explicará el algoritmo para hallar la solución al problema del árbol de expansión mínima de una gráfica mediante computación del ADN, (Vea [11]).

El problema del árbol de expansión mínima es el siguiente:

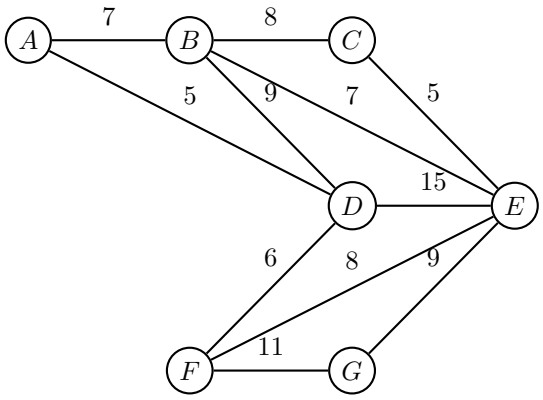
Dada una gráfica $G = (V, E)$, sean $|V| = n$, $|E| = m$, y $w_e \geq 0$ el peso de la arista $e \in E$. Para un árbol $T = (V, E(T))$, con $E(T) \subseteq E$. El problema del árbol de expansión mínima es encontrar el árbol de expansión T de la gráfica G tal que T tiene el mínimo peso.

Para resaltar la importancia de este problema en la realidad, veamos el siguiente ejemplo:

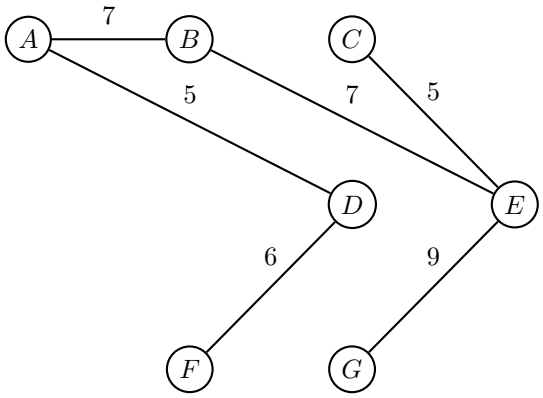
Ejemplo. *El tránsito en Rusia está planificando la construcción de una línea de sistemas de tránsito que conecte 7 zonas de la ciudad (A,B,C,D,E,F,G). Donde los kilómetros entre ellas son los pesos de las aristas. Cada kilómetro de construcción le costará al tránsito 4 millones.*

El tránsito desea acortar la distancia de la traslación entre las 7 zonas pero sin gastar demasiado presupuesto.

La siguiente gráfica muestra la distribución de las ciudades y la distancia entre ellas.



El árbol de expansión mínima tiene peso 39, es decir se construirán 39 kilómetros y su costo será de 156 millones. El árbol queda como sigue:



Para este problema primero se codifican los vértices de la gráfica de manera aleatoria, enseguida se hace la codificación de las aristas mediante enlaces de los vértices, cuando se trabaja con una gráfica ponderada, hace una nueva codificación para identificar los pesos de las aristas. Aquí las secuencias de las aristas estarán representadas por dos componentes, la sucesión de enlaces y la sucesión de aristas.

Después de encontrar la codificación óptima para este problema, se propone construir estructuras gráficas tridimensionales mediante la construcción de bloques de vértices y aristas, como se describe a continuación.

Si un vértice tiene grado k, entonces una molécula de ramificación k es usada para construir este bloque. Los extremos $3'$ de moléculas de ramificación k terminan con extensiones de cadenas simples de longitud 20 y que consisten de la codificación de los vértices mencionada antes. Los bloques de las aristas son justo las codificaciones descritas anteriormente.

Para formar el gráfico todas las moléculas de aristas y todos los bloques de vértices construidos son combinados y sus extremos compatibles permiten formar una doble hebra del ADN. Una vez formados, los bordes están encerrados juntos mediante el sellado de todos los *nicks* abiertos en las hebras de ADN con ADN ligasa. Un *nick* es una discontinuidad en las moléculas de ADN.

El algoritmo para resolver el problema del árbol de expansión mínima se puede resumir como sigue,

Hacer la codificación de la gráfica.

while $(c < C_{max})$ **do**

Síntesis: Producir soluciones candidatas por operadores moleculares.

Separación: Filtro de soluciones no factibles por medidas de laboratorio; donde c es el índice del ciclo, y C_{max} es el índice de ciclo máximo.

end while

if El gráfico tiene n vértices **then**

Mantener sólo aquellos árboles que entran exactamente a n vértices.

Mantener sólo los árboles que entran a todos los vértices de la gráfica al menos una vez.

end if

Seleccionar el árbol que contiene la mayor cantidad de pares de G/C (Guanina y Citosina). Debido a que los pares G/C son químicamente más estables.

Los procesos moleculares del *Paso 2* del algoritmo anterior, pueden describirse como sigue:

1. Se combinan múltiples copias de todos los vértices construyendo bloques con todas las moléculas en un tubo de ensayo, lo que permite que los extremos complementarios logren la hibridación y ligación.

2. Son removidas las estructuras de ADN tridimensional parcialmente formadas con extremos abiertos que no han sido emparejados.

3. Se eliminan por electroforesis en gel las gráficas que son mayores que la gráfica original formada en los pasos anteriores.

Después de realizar este algoritmo se regresa a la codificación de la gráfica para traducir la solución obtenida y conseguir el árbol de expansión mínima de la gráfica.

4.2. La ruta más corta en una gráfica

En las siguientes secciones mostraremos otros problemas que se han solucionado utilizando la computación del ADN, y que pueden ser consultados en [6].

Iniciamos esta sección con un ejemplo que muestra una aplicación práctica del problema de la ruta más corta,

Ejemplo. *En fechas recientes se reservó el área de SEERVADA PARK para paseos y campamentos. No se permite la entrada de automóviles, pero existe un sistema de caminos angostos con curvas*

para tranvías y jeeps conducidos por los guardabosques. El parque contiene un mirador a un hermoso paisaje en la estación T.

La administración quiere deteminar qué ruta desde la entrada del parque hasta la última estación T, es la que tiene la distancia más corta para la operación de los tranvías.

Dada una gráfica ponderada $G = (V, E)$ con $|V| = n$, $|E| = s$ con $\omega_{i,j} \geq 0$ para cada arista $e_{i,j} \in E$. El problema de hallar la ruta más corta de una gráfica es encontrar el camino con menor peso que pasa por todos los vértices de la gráfica.

Para resolver este problema consideremos los símbolos

$$X, \sharp, A_k, B_k, (k = 1, 2, \ldots, n)$$

que son usados para denotar hebras simples de ADN cuya longitud es denotada por $\| \cdot \|$ y por simplicidad tomamos $\| X \| = 10$, además, $\| \sharp \| = \| A_k \| = \| B_k \| = \frac{1}{2} \| X \|$. Las hebras $\omega_{i,j}$ denotan los pesos de las aristas $e_{i,j}$. Así, sean,

$$P = \{\omega_{i,j}, \sharp A_1 B_1, \sharp A_n B_n, \sharp A_k B_k | k = 2, \ldots, n-1\},$$

$$Q = \{\sharp^c, (B_i \omega_{i,j} A_j)^c | e_{i,j} \in E\}, \quad R = \{X\}.$$

El siguiente algoritmo está diseñado para resolver el problema de la ruta más corta de una gráfica.

Paso 1. Elegir todas la posibles rutas que inician en v_1 y terminan en v_n y realizar las siguientes manipulaciones.

Unir(P,Q).

Recocer(P).

Desnaturalizar(P).

Extraer($P, \{\sharp A_1 B_1\}, T_{tmp}, T_{dis}$).

Descartar(P, T_{dis}).

Extraer($T_{tmp}, \{A_n B_n \sharp\}, P, Tdis$).

Después de los pasos anteriores, las hebras simples de ADN en P codifican todas las rutas que inician en v_1 y terminan en v_n.

Paso 2. Cada hebra simple en P denota una ruta factible en la cual la ocurrencia de la subhebra $A_k B_k$ de longitud 20 significa que la ruta pasa por el vértice v_k. Algunas de las rutas factibles pueden no pasar por todos los vértices de la gráfica G. Para evitar éstas, se hacen las manipulaciones siguientes.

for $k = 1 \to n$ **do**

$Extraer(P, \{A_k B_k\}, T, T_{no})$.

$Descartar(P)$.

$Anexar(T_{no}, X)$.

$Unir(P, T, T_{no})$.

end for

Paso 3. Elegir estas hebras de longitud más corta en P.

Sea $k = max_{e_{i,j} \in E} \omega_{i,j}$.

for $m = 1 \rightarrow kn$ **do**

$Seleccionar(P, 20n + 20 + 10m, T)$.

if Detectar(T) es verdadero. **then**

Termina el ciclo.

La ruta más corta es obtenida.

else

El ciclo continúa.

end if

end for

Paso 4. Finalmente usar la operación *Número(T)* para saber el número exacto de aristas en la ruta más corta.

En términos generales el procedimiento anterior puede resumirse así:

- *Paso 1.* A cada vértice de la gráfica se le asigna una sucesión de nucleótidos cuya longitud depende del tamaño de la gráfica.
- *Paso 2.* Se aplican las operaciones de *Recocer* y *Desnaturalizar* al tubo de ensayo P. En este paso, se encuentran todos los caminos de la gráfica representados en P.
- *Paso 3.* Se amplifican las rutas obtenidas en el paso anterior.
- *Paso 4.* Se realizan las operaciones *Amplificar, Extraer* y *Descartar* hasta conseguir que el tubo P esté vacío para librarnos de las moléculas en exceso.
- *Paso 5.* Nuevamente se realiza la operación *Extraer* para tener varios tubos de ensayo que contengan las rutas entre cada par de vértices.
- *Paso 6.* Se aplica la operación *Detectar* para verificar que haya un camino entre cualesquiera dos vértices.
- *Paso 7.* Se utiliza la operación *Seleccionar* a los tubos de ensayo que contienen las rutas entre cada par de vértices hasta encontrar el camino más corto entre cada par de vértices.
- *Paso 8.* Finalmente, *Unir* los tubos de ensayo obtenidos en el paso anterior para formar la ruta más corta entre el vértice inicial y el final.

Al concluir este algoritmo se regresa a la codificación de la gráfica para identificar en ésta a la ruta más corta.

4.3. Coloreado de una gráfica con tres colores

El problema de los tres colores considera colorear una gráfica $G = (V, E)$, de tal manera que cualesquiera dos vértices adyacentes no compartan el mismo color. La solución de este problema es la función $c : V \to \{1, 2, 3\}$, tal que $c(u) \neq c(v)$ para toda arista $(u, v) \in E$.

Consideremos una gráfica $G = (V, E)$ donde $V = \{v_k \mid k = 1, 2, \ldots, n\}$ es el conjunto de vértices y $E = \{e_j \mid j = 1, 2, \ldots, m\}$ es el conjunto de aristas. Primero partimos la gráfica G en dos subgráficas $G_1 = (V_1, E_1)$ y $G_2 = (V_2, E_2)$, tal que $V_1 \cup V_2 = V$ y $V_1 \cap V_2 = \emptyset$, además los conjuntos V_1, V_2 tienen casi el mismo número de elementos, debido a que se eliminan todas las aristas (u, v) tales que $u \in V_1, v \in V_2$, el conjunto de estas aristas eliminadas lo llamaremos C. El proceso de particionar la gráfica se repite recursivamente hasta tener n subgráficas con un sólo vértice.

Para iniciar el procedimiento para hallar la coloración de la gráfica se toma cada una de las n subgráficas y se colorean con algunos de los tres colores disponibles, luego se van uniendo las subgráficas a pares cuidando que no haya conflictos con los colores, en caso de haberlo, se cambia el color para evitar el conflicto; se continúa el procedimiento hasta recuperar por completo la gráfica ya coloreada. Notemos que al ir uniendo las subgŕaficas es necesario reincorporar a la gráfica las aristas del conjunto C, lo que probablemente genere un conflicto de colores.

El algoritmo para resolver el problema en cuestión utiliza las operaciones del ADN definidas al inicio del capítulo y es el siguiente:

for $i = 1 \to n$ **do**
  Síntesis(P_i, color del nodo i).
end for
$f = n$
while $f \neq 1$ **do**
  Hacer múltiples copias de todos los tubos.
  for all i impar **do**
    Ligación(P_i, P_i, P_{i+1}).
    Reetiquetar todos los tubos 1 a $\frac{f}{2}$.
    for $i = 1 \to \frac{f}{2}$ **do**
      for $j = 1 \to E_i$, E_i es el número de aristas de C_i **do**
        Separación($P_i, P_{i_t}, P_{i_f}, \theta_{i_j}$).
        θ_{i_j} es el color conflictivo de la arista e_j, para todo $e_j \in C_i$.
      end for
    end for
  end for
  $f = \frac{1}{2}$

end while

if el tubo está vacío **then**

Responder SI.

else

Responder NO.

end if

En palabras más sencillas el procedimiento que se sigue para resolver este problema puede ser escrito como:

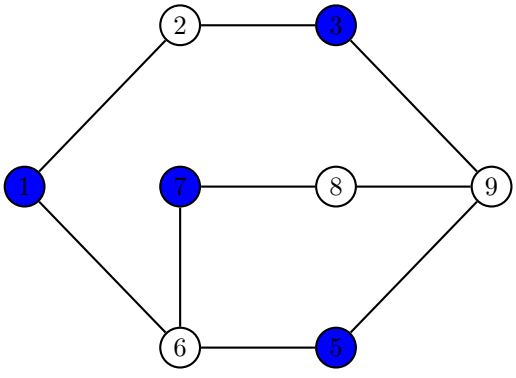
- *Paso 1.* En un ciclo y mediante el proceso de *Síntesis* se obtienen n hebras de ADN para representar los posibles colores que tienen las subgráficas de un sólo vértice.
- *Paso 2.* Se hacen copias de las hebras de ADN de los colores de cada subgráfica.
- *Paso 3.* Se hace la unión de pares de subgráficas mediante el proceso de *Ligación* con las copias obtenidas en el paso anterior.
- *Paso 4.* Después de la ligación, habrá algunos conflictos entre los colores de las subgráficas con las aristas que están en el conjunto C. Estos conflictos son resueltos mediante la operación *Separación*, pues para cada arista se toma el vértice inicial y el final y de cada uno se separan las hebras de acuerdo a cada color.
- *Paso 5.* Se repiten los pasos $2, 3$ y 4 hasta tener la gráfica original coloreada.

4.4. El problema del conjunto dominante

Un conjunto dominante de una gráfica $G = (V, E)$ es un subconjunto $V' \subseteq V$ de vértices tal que para todo $u \in V \setminus V'$ hay un $v \in V'$tal que $(u, v) \in E$.

El problema del conjunto dominante de una gráfica es hallar el conjunto dominante de tamaño mínimo.

Ejemplo. *Consideremos la siguiente gráfica,*



El conjunto dominante de esta gráfica es $\{1, 3, 5, 7\}$*, decimos que tiene tamaño mínimo puesto que si suprimimos un vértice de él, deja de ser un conjunto dominante.*

Consideremos un valor no asignado X determinado por variables binarias $x_n, x_{n-1}, \ldots, x_1$. El rango de los valores de X es desde 0 hasta $2^n - 1$, esto dice que está formado por 2^n valores posibles, cada valor representa un conjunto dominante de la gráfica G. Así, una variable X forma 2^n posibles conjuntos dominantes. Un bit x_i representa el i-ésimo vértice en G. Si el vértice i está en un conjunto dominante entonces x_i toma el valor de 1.

El algoritmo para resolver el problema del conjunto dominante es:

Sea T_0 un tubo que contiene las codificaciones de todos los conjuntos dominantes.
for all i=0:n, n es el número de vértices en G **do**
  Extraer(T_0, x_i^1, P, R).
  for v_j adyacente a v_i **do**
    Extraer(R, X_j^1, R^+, R^-).
    Unir(T_0, R^+).
  end for
  Descartar(R).
end for
for all $i = 0$ hasta $i = n - 1$ **do**
  for $j = i$ hasta 0 **do**
    Extraer$(T_j, x_{i+1}^1, T_{j+1}^{on}, T_{j+1}^{of})$.
    Unir(T_{j+1}, T_{j+1}^{on}).
  end for
end for
for $k = 1 \rightarrow n$ **do**
  if Detectar(T_k) es verdadero **then**
    Lee(T_k) y el algoritmo termina.
  end if
end for

4.5. Simulación en computadora

Como comentamos al inicio del capítulo, estos algoritmos que se han explicado, sólo han sido desarrollados en un laboratorio de química, sin embargo se han intentado programar en una computadora. En esta sección daremos algunas referencias acerca de lo que se ha hecho para obtener una simulación en computadora de algoritmos de ADN.

4.5.1. Algoritmo de Adleman

Uno de los trabajos que se han realizado en este aspecto, es la simulación del algoritmo de Adleman para hallar una ruta hamiltoniana en una gráfica dirigida, el cual fue desarrollado por Sanjeev Baskiyar [4]. En el cual, se eligió implementar el algoritmo de Adleman debido a que es más fácil de implementar y varias de otras aplicaciones se desprenden de este modelo. A continuación explicaremos brevemente en que consistió esta simulación.

El algoritmo de Adleman es el siguiente: Sea $G(V, E)$ una gráfica dirigida con V el conjunto de vértices y E el conjunto de aristas, llamaremos v_{in} y v_{out} al vértice inicial y final respectivamente, llamamos ruta hamiltoniana a una ruta que inicie en v_{in}, y termine en v_{out} y que recorre cada vértice exactamente una vez. El algoritmo es el siguiente,

1. Generar rutas aleatorias en $G = (V, E)$.
2. Eliminar rutas que no inician en v_{in} y termina en v_{out}.
3. Eliminar las rutas que no tienen $|V|$ vértices.
4. Eliminar rutas que visitan algún vértice más de una vez.
5. Si hay alguna ruta que no haya sido eliminada, regresar *verdadero*, de otra forma regresar *falso*.

Para implementar el Paso 1 del algoritmo, cada vértice del gráfico se codificó como una cadena aleatoria de 20 nucleótidos o una secuencia de 20 letras de ADN. Luego, por cada arista de la gráfica, se creó una secuencia de ADN cuya segunda mitad codificó el vértice de inicio, y la primera mitad el vértice final. Mediante el uso de complementos de los vértices como férulas, se ligaron secuencias de ADN correspondientes a bordes compatibles, formando moléculas de ADN que codifican caminos aleatorios a través de la gráfica.

Para implementar el Paso 2, la salida del Paso 1 se amplificó por PCR (Reacción en cadena de la polimerasa) es una técnica que consiste en la amplificación in vitro de un fragmento de ADN específico. Para llevar a cabo el experimento de amplificación es necesario conocer, al menos parcialmente, la secuencia del fragmento a amplificar (un gen, una parte de un gen, una región no codificadora,...). Básicamente, se trata de replicar una y otra vez un mismo fragmento de ADN y, para ello, debemos realizar in vitro lo que hacen las células en vivo para replicar su ADN. Por lo tanto, sólo aquellas moléculas que codifican las rutas que comenzaron con v_{in} el vértice inicial y terminaron en v_{out} el vértice final fueron amplificadas.

Para implementar el Paso 3, se utilizó una técnica llamada electroforesis en gel, que separa hebras de ADN por longitud. Las moléculas se colocan encima de un gel húmedo, al cual se le aplicó un campo eléctrico, y las moléculas comenzaron a moverse, las moléculas más grandes se

mueven más lentamente que las más pequeñas. Así, después de un cierto tiempo, las moléculas se separan según el tamaño.

El paso 4 se realizó mediante el uso iterativo de un proceso llamado purificación por afinidad. Este proceso permite filtrar hebras con una subsecuencia dada que codifican un vértice v. Después se sintetizan hebras complementarias a v y se adjuntan a cuentas magnéticas. Las secuencias que tienen a v y a su secuencia complementaria, se retienen, y las demás se filtran.

En el Paso 5, se comprobó la presencia de una molécula que codifica una ruta hamiltoniana.

En la siguiente subsección se explicará como se implementó este algoritmo mediante software computacional.

Software

El sistema de simulación tiene tres módulos: Codificación del gráfico, Combinador de ADN y Filtros de ruta. La codificación de la gráfica a su vez tiene dos módulos de software: el analizador / el paquete creador de ADN y el replicador de ADN. El analizador / creador de paquetes de ADN analiza el archivo de entrada y crea una matriz de paquetes de ADN con cada paquete que representa un nodo en la gráfica de entrada. Envía el tamaño del gráfico, $|V|$, al combinador de ADN y la matriz de paquetes de ADN para el replicador de ADN.

El módulo combinador de ADN tiene también dos módulos, el generador de números aleatorios y el motor del ADN. El generador de ADN genera números aleatorios dentro del conjunto acotado $(0, \ldots, |V| - 1)$. Cuando el motor del ADN requiere un paquete de ADN que identifica a un nodo determinado n. El motor del ADN imita las operaciones de combinación del ADN y determina posibles rutas que podrían ser hamiltonianas. Las rutas posibles archivan algunas de las cuales pueden ser rutas hamiltonianas. Los filtros de rutas que tienen índices duplicados, cuyas longitudes no son $|V|$ y que no comienzan en v_{in} y terminan en v_{out}.

4.5.2. Optimización de la ruta de un elevador

Otro problema resuelto con computación de ADN que se ha intentado simular en una computadora, es el problema de optimizar las rutas de un edificio con N pisos y M elevadores. Este problema fue simulado y estudiado por M.S.Muhammad y sus colaboradores, [13].

Explicaremos brevemente en que consiste este problema de optimización. Se analizarán las posiciones de dos elevadores en un edificio de 6 pisos. Si la posición de cada elevador en el piso $1, 2, 3, 4, 5$, y 6 son representados como los nodos v_1, v_2, v_3, v_4, v_5, y v_6 respectivamente, cada una de las combinaciones de las rutas de viaje del elevador pueden ser representadas mediante una gráfica ponderada. Al mismo tiempo, las aristas de la gráfica representan así las trayectorias de viaje entre pisos.

El algoritmo para resolver este problema con computación de ADN es el siguiente,

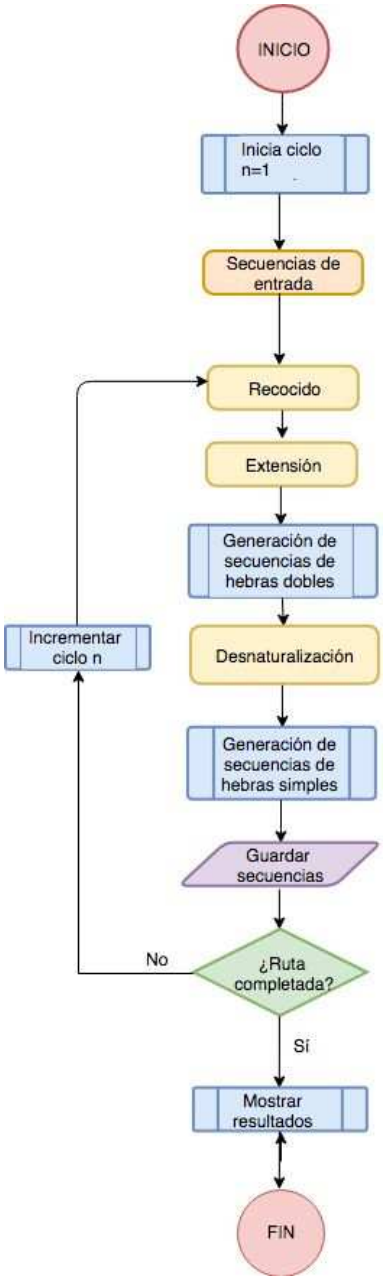


Figura 4.1: Algoritmo de optimización de rutas

1. Se representan las rutas del elevador en un gráfico ponderado.
2. El gráfico se transforma para representar los nodos de inicio, intermedio y final y para diferenciar los nodos de diferentes combinaciones de rutas.
3. Se asigan secuencias de ADN a cada nodo.
4. Se sintetizan las combinaciones de rutas de modo que la longitud de secuencia de oligonucleótidos de ADN será directamente representados por el peso entre los nodos.
5. Se generan todas las combinaciones de rutas posibles.
6. Mediante reacción en cadena polimerasa se amplifican las rutas posibles.
7. Se aplica electroforesis en gel para separar las rutas de acuerdo a su longitud, y se selecciona la ruta menor que será la óptima.

La implementación de la computación del ADN implica el diseño de secuencias de oligonucleótidos para representar tanto las entradas como las salidas del problema. A medida que el problema aumenta, la complejidad del diseño de los oligonucleótidos se complica. Para ayudar en el diseño, se desarrolló un programa de simulación simple que puede simular el proceso de ADN esperado durante la etapa de cálculo.

Los autores de este trabajo desarrollaron un software en la plataforma Microsoft Visual Basic que es capaz de simular los procesos físicos del ensamblaje de las hebras de ADN para formar la doble hélice y el PCR (Reacción en cadena polimerasa) de la computación. El diagrama de flujo del programa de software diseñado se muestra en la Figura 4.1.

Conclusiones

En este trabajo de tesis se han presentado algunos conceptos básicos de teoría de códigos. También se mencionan algunas definiciones y teoremas básicos de teoría de campos y de anillos de polinomios que permiten construir códigos con propiedades especiales.

Principalmente, en esta tesis se hizo un análisis a detalle del artículo de Abualrub [2], en el que se dan condiciones para los códigos cíclicos sobre $GF(4)$ que permiten dar una mejor codificación para los algoritmos de ADN. En este trabajo se describen los elementos básicos que se deben considerar para crear códigos que sea útiles para representar cadenas de ADN. Se presentan demostraciones detalladas de resultados que permiten construir códigos aditivos apropiados para modelar cadenas de ADN, a fin de implementar computacionalmente algoritmos de ADN. Después de explicar detalladamente la teoría necesaria para la construcción de los códigos aditivos que nos interesan, se dan ejemplos de cómo aplicar esta teoría para la construcción de códigos cíclicos de longitud 7.

Finalmente se describen algunas aplicaciones de la teoría en la resolución de problemas de matemáticas discretas; se presentan ejemplos específicos que han sido resueltos con la computación del ADN.

En el futuro sería interesante crear algoritmos computacionales que empleen los códigos que aquí generamos, y utilizarlos para resolver problemas complejos. También habría que ver si existen códigos con mayor distancia que los que aquí se construyeron, y ver si dichos códigos podrían ser útiles en la computación de ADN. En este trabajo solamente se dieron ejemplos de algoritmos de ADN que se utilizaron para resolver problemas de matemáticas discretas, sin embargo, una pregunta interesante que surge de inmediato es ¿Pueden utilizarse los algoritmos de ADN para solucionar problemas en otras áreas de las matemáticas? La respuesta a esta pregunta puede dar como resultado un interesante trabajo futuro.

Bibliografía

[1] Taher Abualrub, Ali Ghrayeb, and Xiang Nian Zeng. A special class of additive cyclic codes for DNA computing. In *Adaptive and Natural Computing Algorithms*, pages 284–287, Vienna, 2005. Springer Vienna.

[2] Nian Zeng X. Abualrub T., Ghrayeb A. Construction of cyclic codes over $GF(4)$ for DNA computing. *Franklin Inst*, 343(4-5):448–457, 2006.

[3] Leonard M. Adleman. Molecular computation of solutions to combinatorial problems. *Science*, 266(11):1021–1024, November 1994.

[4] Sanjeev Baskiyar. Simulating DNA computing. In *Proceedings of the 9th International Conference on High Performance Computing*, HiPC '02, pages 411–419, London, UK, UK, 2002. Springer-Verlag.

[5] A. R. Calderbank, E. M. Rains, P. M. Shor, and N. J. A. Sloane. Quantum error correction via codes over GF(4). *IEEE Transactions on Information Theory*, 44(4):1369–1387, July 1998.

[6] Hossein Eghdami and Majid Darehmiraki. Application of DNA computing in graph theory. *Artif Intell Rev*, 38:223–235, 2012.

[7] John B. Fraleigh. *Álgebra abstracta*. Sistemas Técnicos de Edición, 1988.

[8] N. Herstein, I. *Abstract Algebra*. Prentice Hall, 1996.

[9] Jon-Lark Kim and Vera Pless. Designs in additive codes over GF(4). *Designs, Codes and Cryptography*, 30:187–199, 2003.

[10] James L. Massey. Reversible codes. *Information and Control*, 7:369–380, 1964.

[11] Xikui Liu, Yan Li, and Jin Xu. Solving minimum spanning tree problem with DNA computing. *Journal of Electronics (China)*, 22(2):112–117, 2005.

[12] Olgica Milenkovic and Navin Kashyap. On the design of codes for DNA computing. *Lecture Notes in Computer Science*, 3969:100–119, 2006.

[13] M.S. Muhammad, Masniah Masra, Kuryati Kipli, and Nurdiani Zamhari. A simulation software for DNA computing algorithms implementation. *International Journal of Computer and Information Engineering*, 4(12):1819–1826, 2010.

[14] A. Neubauer, J. Freudenberger, and V. Kuhn. *Coding Theory: Algorithms, Architectures and Applications*. Wiley, 2007.

[15] Somnath Tagore, Saurav Bhattacharya, Md Ataul Islam, and Md Lutful Islam. DNA computation: Applications and perspectives. *Journal of Proteomics and Bioinformatics*, 3(7), 2010.