Situación Problema: Fundamentos del Algebra Lineal

Ricardo Salgado Benítez - A01282489
Tecnológico de Monterrey
Docente: Jesús Jorge Armenta Segura

12 de septimbre de 2025

1 Resumen

En este reporte, se presenta un método para llevar a cabo códigos de Hamming mediante herramientas de álgebra lineal, a diferencia de los métodos tradicionales que recurren a operadores lógicos. Las herramientas en cuestión son el campo de Galois {0, 1}, las matrices y la multiplicación de estas últimas. La primera sección de este reporte, explica el problema que se busca resolver mediante los códigos de Hamming. La segunda sección aborda las herramientas refereidas anteriormente del álgebra lineal. La siguiente sección es el núcleo del reporte y explica el método que fue usado para implementar los códigos de Hamming. Posteriormente, se abordan las conclusiones de este reporte. Finalmente, se incluyen anexos que contienen una implementación del método del reporte tanto en *Google Colab* como en *GitHub*, además de una explicación del método usado para acelerar la multiplicación de matrices.

2 Introducción al Problema

De acuerdo con el mismo Hamming (1950), el objetivo principal detrás de los códigos que hoy llevan su nombre era evitar errores singulares en lo que hoy se conoce como computadoras. En particular, este autor desctaca este problema en el contexto de comunicación entre dos computadoras, se da el ejemplo de comunicaciones entre computadoras de la compañía *Bell Telephone Laboratories*, donde un error fue observado en 2 a 3 de cada 8900 procesos de comunicación diarios. Aunque estos numeros sean pequeños, Hamming destaca el hecho que en las computadoras de la epoca este era un fallo catastrófico que frencuentemente causaba un alto completo del programa en ejecución.

Si bien, los avances en *hardware* y *sofware* han permitido que los equipos de cómputo actuales sean más resilientes tanto a los errores individuales en el código como a los fallos en el programa. Sin embargo, la función de evitar errores sigue siendo crucial para todos los dispositivos digitales, como bien lo indica Sanderson (2013). Además, en el curso de 7 décadas de avance tecnológico, han surgido un amplia gama de otros códigos que son aún más eficientes que los códigos de Hamming a la hora de detectar errores. De estos, sin duda los más destacables son los *códigos de Reed-Solomon* y los llamados *códigos Turbo* (Referencia).

Pese a esto, los códigos de Hamming presentan una buena base para ser el proyecto de una clase de con el nombre de *Fundamentos del Álgebra Lineal*, aunque ciertamente no la única aplicación de esta área de las matemáticas en la corrección de errores. Si bien, Hamming solo se limita a presentar la manera de llevar a cabo su método y no a implementarlo de ninguna especifica, en el curso de la historia de la computación ha sido realizarlo mediante operadores lógicos (Sanderson, 2020). Esta última implementación resulta en una lógica computacional extremadamente sencilla, si algo poco elegante matemáticamente hablando. Afortunadmente, el álgebra lineal posee dos herramientas centrales que permitirán traducir estos operadores logicos en operaciones matemáticas

bien definidas. Estas herramientas serán el tema central de la siguiente sección y son: los *Campos de Galois* y la *Multiplicación de Matrices*.

3 Fundamentos del Álgebra Lineal

3.1 Campo de Galois {0, 1}

Incluso en 1950, cuando Hamming definió su código, el lenguaje computacional era binario, como lo sigue siendo en la actualidad. Como un lector astuto podrá notar el campo binario es distinto del campo de los reales, de los enteros y de los complejos y, por lo tanto, requiere su propia definición. Dado que este será un campo finito con solo 2 elementos (0 y 1), llamase este *Campo de Galois {0, 1}* en honor al siempre jóven matemático fránces, Évariste Galois. Una última cuestión es que, debido a que solo existen 2 elementos en este campo, es relativamente práctico llevar a cabo pruebas exhaustivas para demostrar los axiomas del mismo como se desarrolla a continuación (si bien vairas propiedades se justifican mediante la herencia de los enteros).

Sea $Z_2 = 0, 1$ el campo de los números binarios con los operadores de suma y producto definidos como:

- $a + b = 0$ si $(a + b)/2 = 0$ (división de números reales), $a + b = 1$ de lo contrario.
- $a \star b =$ si a o b es 0 y $a \star b = 1$ en caso contrario.

Para demostrar que Z_2 es un campo es necesario demostrar que ambos operadores cumplen con conmutatividad y asocitavidad, así como que tengan un neutro y un inverso.

- CONMUTATIVIDAD ADITIVA

$$0 = 0 + 0 = 0 = 0 + 0$$

$$1 + 0 = 1 = 0 + 1$$

$$1 + 1 = 0 = 1 + 1$$

  Por lo tanto, los elementos de la adición en Z_2 son conmutativos.

- ASOCIATIVIDAD ADITIVA

$$0 + (0 + 0) = 0 + 0 = (0 + 0) + 0$$

$$0 + (0 + 1) = 0 + 1 = (0 + 0) + 1$$

$$0 + (1 + 0) = 0 + 1 = 1 + 0 = (0 + 1) + 0$$

$$0 + (1 + 1) = 0 + 1 = (0 + 1) + 1$$

$$1 + (0 + 0) = 1 + 0 = (1 + 0) + 0$$

$$1+(0+1)=1+1=(1+0)+1$$
$$1+(1+0)=1+1=0=0+0=(1+1)+0$$
$$1+(1+1)=1+1=(1+1)+1$$

Por lo tanto, los elementos de la adición en Z_2 son asociativos.

- NEUTRO ADITIVO

$$0+0=0$$
$$1+0=1$$

Por lo tanto, 0 es el neutro aditivo de Z_2.

- INVERSO ADITIVO

$$0+0=0$$
$$1+1=0$$

Por lo tanto, en Z_2 existe un elemento que hace que cada uno de sus elementos se vuelva igual al neutro aditivo (0).

- CONMUTATIVIDAD MULTIPLICATIVA

$$0\star 0=0=0\star 0$$
$$0\star 1=0=1\star 0$$
$$1\star 1=1=1\star 1$$

Por lo tanto, los elementos de la multiplicación en Z_2 son conmutativos.

- ASOCIATIVIDAD MULTIPLICATIVA

$$0\star(0\star 0)=0\star 0=(0\star 0)\star 0$$
$$0\star(0\star 1)=0\star 0=0=0\star 1=(0\star 0)\star 1$$
$$0\star(1\star 0)=0\star 0=(0\star 1)\star 0$$
$$0\star(1\star 1)=0\star 1=(0\star 1)\star 1$$
$$1\star(0\star 0)=1\star 0=0=0\star 0=(1\star 0)\star 0$$
$$1\star(0\star 1)=1\star 0=0=0\star 1=(1\star 0)\star 1$$
$$1\star(1\star 0)=1\star 0=(1\star 1)\star 0$$
$$1\star(1\star 1)=1\star 1=(1\star 1)\star 1$$

Por lo tanto, los elementos de la multiplicación en Z_2 son asociativos.

- NEUTRO MULTIPLICATIVO

$$1 \star 0 = 0$$

$$1 \star 1 = 1$$

Por lo tanto, existe un elemento (el nuetro aditivo, 1) de Z_2 que multiplicado por un elemento resulta en el mismo elemento.

- INVERSO MULTIPLICATIVO

$$1 \star 1 = 1$$

0 no tiene inverso multiplicativo, sin embargo 0 es un elemento nulo en Z_2 de la misma manera que los es en el campo de los reales (es decir, $0 \star x = 0, \forall x \in R \vee \forall x \in Z_2$).

Por lo tanto, existe un inverso multiplicativo en Z_2 tal que todos sus elementos no nulos multiplicados por este resulten iguales al *neutro multiplicativo*.

Por último, es necesario demostrar que estos operadores se relacionan mediante los siguientes axiomas de distribuitividad.

- AXIOMAS DE DISTRIBUITIVIDAD

$$0 \star (0+0) = 0 \star 0 = 0 = 0 + 0 = 0 \star 0 + 0 \star 0$$
$$0 \star (1+0) = 0 \star (0+1) = 0 \star 1 = 0 = 0 + 0 = 0 \star 0 + 0 \star 1 = 0 \star 1 + 0 \star 0$$
$$0 \star (1+1) = 0 \star 0 = 0 = 0 + 0 = 0 \star 1 + 0 \star 1$$
$$1 \star (0+0) = 1 \star 0 = 0 = 0 + 0 = 1 \star 0 + 1 \star 0$$
$$1 \star (1+0) = 1 \star (0+1) = 1 \star 1 = 1 = 0 + 1 = 0 \star 0 + 1 \star 1 = 1 \star 1 + 0 \star 0$$
$$1 \star (1+1) = 1 \star 0 = 0 = 1 + 1 = 1 \star 1 + 1 \star 1$$

Dado que Z_2 con la definición anterior cumple con todos estos axiomas se concluye que es un campo.

Al Z_2 ser un campo, se podrá incorpar junto con la siguiente herramienta.

3.1.1 Operadores lógicos en el Campo de Galois {0, 1}

El operador lógico binario *XOR* recibe 2 entradas booleanas (que para propósitos de esta explicación interpretense 0 como falso y 1 para verdadero) y regresa verdadero (1) si solo una de las entradas es verdadera y falso (0) de lo contrario.

El operador lógico binario *AND* recibe 2 entradas booleanas y regresa verdadero (1) si ambas son verdaderas o falso (0) de lo contrario.

De acuerdo con la interpretación de falso como 0 y verdadero como 1, se puede observar que *XOR* tiene el mismo comportamiento que la adición en Z_2, mientras que *AND* comparte su comportamiento con la multiplicación en Z_2. Es decir, para $A, B, C \in Z_2$:

- A XOR $B = C \iff A + B = C$

  Proof.

  $$0 \text{ XOR } 0 = 0 = 0 + 0$$
  $$0 \text{ XOR } 1 = 1 = 0 + 1$$
  $$1 \text{ XOR } 0 = 1 = 1 + 0$$
  $$1 \text{ XOR } 1 = 0 = 1 + 1$$

  □

- A AND $B = C \iff A \star B = C$

  Proof.

  $$0 \text{ AND } 0 = 0 = 0 \star 0$$
  $$0 \text{ AND } 1 = 0 = 0 \star 1$$
  $$1 \text{ AND } 0 = 0 = 1 \star 0$$
  $$1 \text{ AND } 1 = 1 = 1 \star 1$$

  □

Estas propiedades serán cruciales para traducir los códigos de Hamming con operadores binarios al lenguaje del Álgebra Lineal.

3.2 Matrices

Para $m, n \in Z^+$, una matriz $n \times m$ (llamese A) es un ordenamiento rectangular de elementos de un campo F con m columnas y n filas:

$$A = \begin{pmatrix} A_{11} & A_{12} & \cdots & A_{1m} \\ A_{21} & A_{22} & \cdots & A_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ A_{n1} & A_{n2} & \cdots & A_{nm} \end{pmatrix}$$

Sea A_{jk} la entrada en la fila j y la columna k (Axler, 2025).

Dado que se demostró que Z_2 es un campo, la defínción de matriz implica sus elementos pueden pertenecer a Z_2. Por ejemplo, B es una matriz 3×2 sobre Z_2.

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

Existen diversas operaciones que pueden ser llevadas a cabo con matrices, por ejemplo *adición de matrices* y *multiplicación escalar*. Sin embargo para los propósitos de este reporte, la única operación relevante es la multiplicación de matrices.

3.3 Multiplicación de Matrices

De acuerdo con Axler (2025), la múltiplicación de matrices se define de la siguiente manera:

Sea A una matriz $m \times n$ y B una matriz $n \times p$. Entonces AB es una matriz $m \times p$, donde cada elemento se define de la siguiente manera:

$$(AB)_{jk} = \sum_{r=1}^{n} A_{jr} B_{rk}$$

$(AB)_{jk}$ se calcula multiplicando las entradas correspondientes de la fila j de A y la columna k de B y sumando los resultados de todas estas multiplicaciones.

Notese que suma (adición) y multiplicación en el último párrafo se refieren a la manera en la que estas operaciones están definidas en el campo correspondiente dado.

Por ejemplo, sean A y B matrices 1×5 y 5×2 (respectivamente) sobre el campo Z_2:

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

Donde AB es una matriz 1×2 sobre Z_2 y sus elementos son:

$$(AB)_{11} = 0 \star 1 + 1 \star 0 + 1 \star 0 + 0 \star 1 + 0 \star 0 = 0 + 0 + 0 + 0 + 0 = 0$$

$$(AB)_{12} = 0 \star 1 + 1 \star 0 + 1 \star 1 + 0 \star 1 + 0 \star 1 = 0 + 0 + 1 + 0 + 0 = 1$$

4 Solución Propuesta

Las implementaciones en álgebra lineal de los códigos de Hamming explicadas a continuación fueron desarrolladas por el autor a partir de los métodos (que no son de álgebra lineal) explicados por Sanderson (2025), por lo tanto, puede que se rompan con algunas convenciones. A continuación, se llevará a cabo como implementar álgebra lineal tanto al momento de codificar una mensaje de acuerdo a los códigos de Hamming como al momento de descubrir la posición del error.

4.1 Códificación por códigos de Hamming

4.1.1 Método tradicional

Todo implementación de los *Códigos de Hamming* ocupa aumentar el tamaño de la matriz inicial (a partir de ahora llamada *mensaje*) para poder llevarse a cabo. Estos elementos que son agregados se conocen como *bits de pariedad* o

generalizando un poco *elementos de pariedad*. Estos elementos de pariedad son tales que aseguren que el número de elementos positivos (1) en ciertas posiciones sea par. Estas ciertas posiciones, así como la posición de los elementos de pariedad, están dadas por la siguiente tabla:

Posiciones de los elementos de pariedad	Posiciones de los elementos checados
1	1, 3, 5, 7, 9, 11, 13, 15, 17, ...
2	2, 3, 6, 7, 10, 11, 14, 15, 18, ...
3	4, 5, 6, 7, 12, 13, 14, 15, 20, ...
4	8, 9, 10, 11, 12, 13, 14, 15, 24, ...
⋮	⋮

En otras palabras, los elementos de pariedad siempre están en una posición $2^i, \forall i \in Z$ y, si se considera la representación en binario de las posiciones que checan, los elementos de pariedad checan todas las posiciones que tienen activado el *bit* que corresponde a la posición del elemento de pariedad.

Por ejemplo la representación binario para los primeros 7 números es la siguiente (las columnas indican el valor del bit que esta activado):

Número	1	2	4
1	1	0	0
2	0	1	0
3	1	1	0
4	0	0	1
5	1	0	1
6	0	1	1
7	1	1	1

Por esto es que el elemento de pariedad en posición 1 revisa los elementos en las posiciones 1, 3, 5 y 7, la posición 2 revisa 2, 3, 6 y 7, etc.

Como se dijo al inicio el objetivo es que todas las posiciones que sean revisadas por un número tengan un número par de 1s. Es decir, sea $S = \{s_1, ..., s_n\}$ el conjunto de elementos en Z_2 que se revisen, entonces:

$$0 = s_1 + ... + s_n$$

Ahora, codificar un mensaje resulta bastante sencillo, ya que solo hay que agregar elementos en las posiciones indicadas hasta que la posición indica sea más grande que el tamaño del *mensaje* para después llevar a cabo las pruebas indicadas. Por ejemplo, la matriz A 1×4:

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

Se codificaría como la siguiente matriz A' 1×8:

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

4.1.2 Método mediante Álgebra Lineal

De la misma forma que en el método tradicional, es necesario agregar elementos extra al *mensaje*, que se puede representar como una matriz sobre Z_2 donde cada elemento corresponde a un *bit*. Estos elementos deberán ocupar las posiciones estarán en el siguiente conjunto $P = \{0\} + \{2^i | i = 0, ..., \lfloor \log_2(m) \rfloor\}$ donde m es el largo del *mensaje*. La posición 0 no tiene utilidad para este método, pero necesita estar ocupado para que esto funcione. Como en el otro, método estas posiciones pueden ser ocupadas por 0 inicialmente.

El método tradicional "selecciona", por así decirlo, las posiciones que tienen el bit correspondiente activado. Dentro de Z_2, la multiplicación se puede considerar como una especie de selección. Esto debido a las propiedades que tiene multiplicar por 0 en Z_2 (y en los enteros también) de mandar cualquier número a 0. Por lo tanto, multiplicar por 1 se puede considerar como seleccionar un número y por 0 como deseleccionarlo. En realidad, esta equivalencia depende del uso que se le de a este procedimiento, pero dado que la siguiente operación va a ser una adición y que el nuetro aditivo es 0, este razonamiento es válido.

Una vez ya se han seleccionado todos las posiciones correspondientes a cada elemento de pariedad (mediante multiplicación), los elementos en estas posiciones se deben de sumar (adición de Z_2). Sea el resultado de está suma s y p el valor del elemento de pariedad,

- Si $s = 0$, $p = 0$ es necesario (para asegurar un número par de 1s)
- Si $s = 1$, $p = 1$ es necesario.

Por lo tanto, $s = p$ para todos los casos.

Lo descrito anteriormente no es más que una múltiplicación de matrices. La mayor dificultad consiste en generar una matriz que "seleccione" las posiciones indicadas. Esta matriz debe de multiplicar por 1 el valor de pariedad actual y todos los otros elementos en los que esta activo el bit, dejando este valor en su posición correspondiente. Cuando la posición del resultado no corresponde a un índice de pariedad, el único valor que se debe de multiplicar por 1 es el valor actual.

Los requierimientos anteriores se cumplen con la matriz cuadrada D $n \times n$, donde n es el largo del mensaje, y sea que D es generada por una función Φ a partir de la matriz del mensaje (con 0s como elementos de pariedad). La primera columna de D consiste en 0s (opcionalmente, se puede que el elemento en la primer posición de la matriz resultante de multiplicar el mensaje con D sea la suma en Z_2 de todos los elementos de la matriz para que esta sea capaz de identificar 2 errores en el mensaje, pero esto va más allá de los límites de la actividad). Las columnas cuyos números corresponden a las posiciones de los índices de pariedad contienen 1s en los lugares en los que esta activado el bit correspondiente. Las columnas con valores que no son elementos de pariedad tienen un único 1 en el renglón que corresponde a su número de columna.

Por ejemplo, retomando la matriz A, sea A_{ext} la matriz con las posiciones de pariedad agregadas como 0,

$$A' = A_{ext}\Phi(A_{ext}) = \begin{pmatrix}0 & 0 & 0 & 1 & 0 & 0 & 0 & 1\end{pmatrix} \begin{pmatrix}0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0\\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0\\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0\\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0\\ 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1\end{pmatrix}$$

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

El resultado es el mismo que anteriormente, por lo que el método funciona (en el código, se llevan a cabo pruebas con *PyTest* que comprueban el funcionamiento de este método con otras matrices)

Esté método también funciona para mensajes codificados cuyo largo no sea una potencia de 2. Por ejemplo, sea B_{ext} una matrix 1×6 que representa el mensjae codificado con los elementos de pariedad como 0s,

$$B' = B_{ext}\Phi(A_{ext}) = \begin{pmatrix}0 & 0 & 0 & 1 & 0 & 1\end{pmatrix} \begin{pmatrix}0 & 0 & 0 & 0 & 0 & 0\\ 0 & 1 & 0 & 0 & 0 & 0\\ 0 & 0 & 1 & 0 & 0 & 0\\ 0 & 1 & 1 & 1 & 0 & 0\\ 0 & 0 & 0 & 0 & 1 & 0\\ 0 & 1 & 0 & 0 & 1 & 1\end{pmatrix}$$

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

El lector puede comprabar que este resultado es válido con respecto al método tradicional

4.2 Identificación de errores en códigos de Hamming

4.2.1 Mediante Software

Una vez el *mensaje* ha sido codificado, la posición del error puede ser identificada muy fácilmente mediante la siguiente línea de código en *Python* (las primeras dos no son contadas, ya que su utilidad es importar librerías):

```
from functools import reduce
import operator as operaciones
reduce(op.xor, [i for i, bit in enumerate(message) if bit])
```

Debido a que la operación *XOR*, trabaja bit por bit (o, elemento por por elemento), esto es equivalente ejecutar *XOR* en cada bit para cada una de las posiciones activas (es decir, con un 1). Por ejemplo para la matriz A' estas líneas de código serían equivalentes a:

$$\begin{array}{cccc} & 0 & 1 & 1 \\ & 1 & 0 & 0 \\ \oplus & 1 & 1 & 1 \\ \hline & 0 & 0 & 0 \end{array}$$

Donde $\oplus$ denota `reduce(op.xor, ...)`.

Por supuesto el resultado actual es 0 (en binario), pero si se cambia el valor de la columna 6, el resultado sería

$$\begin{array}{cccc} & 0 & 1 & 1 \\ & 1 & 0 & 0 \\ & 1 & 1 & 0 \\ \oplus & 1 & 1 & 1 \\ \hline & 1 & 1 & 0 \end{array}$$

El resultado es exactamente la representación binaria de 6. Como dije es una implementación bastante sencilla.

Para traducir este método al lenguaje del álgebra lineal, es necesario preguntarse sobre la existencia de una herramienta dentoro que siga estos pasos. Siendo "estos pasos": seleccionar las posiciones que tengan 1 y agregar los resultados de sus posiciones mediante *XOR*.

4.2.2 Mediante Álgebra Lineal

Como se vío anteriormente, la múltiplicación en Z_2 es equivalente a *AND* y la adición a *XOR*. La última operación de una múltiplicación de matrices es una adición de todos los resultados de las multiplicaciones. Y, de cierta manera, las multiplicaciones en Z_2 actuan como un selector dejando pasar los elemento apareados con un 1 y deteniendo a los que van con 0. Lo único que queda es pasar las representaciones binarias de las posiciones a la multiplicación de matrices.

Por ejemplo, sea A una matriz 1×4 en Z_2 ya codificada,

$$A = \begin{pmatrix} 1 & 1 & 1 & 1 \end{pmatrix}$$

Entonces, sea B una matriz en Z_2 con las 4 posiciones de A representadas en binario por los elementos de cada renglón:

$$B = \begin{pmatrix} 0 \text{ en binario} \\ 1 \text{ en binario} \\ 2 \text{ en binario} \\ 3 \text{ en binario} \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{pmatrix}$$

Entonces, AB otorgará el resultado deseado de la representación binaria de la posición alterada.

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

Cuando no hay alteraciones, el resultado es 0, dado que la posición no tiene un uso (al menos, en esta vesión del código de hamming), es aceptable que se indique esta posición. En caso de que A fuera alterada, el resultado sería distinto:

$$(A')B = \begin{pmatrix} 1 & 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 1 \end{pmatrix}$$

Siendo este resultado la representación binaria de 2 que es la posición donde ocurrió la alteración.

Para evitar este ejercicio mental de interpretar la representación binaria, el resultado anterior se puede multiplicar por otra matriz que otorgue un número en decimal, solo habría que asegurar que cada elemento binario se multiplique con el número decimal que representa para después sumarse. Una complicación de esta operación es que AB es una matriz sobre Z_2 que se estaría multiplicando con una matriz sobre Z. Para aliviar este problema es necesario pasar AB por una función (llamese Λ) que mapee cada uno de sus elementos a Z:

- $0 \in Z_2$ se convertiría en $0 \in Z$
- $1 \in Z_2$ se convertiría en $1 \in Z$

De esta manera, sea C una matriz vertical 2×1 (claro que esto varía dependiendo del número de bits necesarios para representar el elemento en la última columna de A) donde $A_{i1} = 2^(i-1)$, entonces

$$\Lambda(AB)C = \begin{pmatrix} 0 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 2 \end{pmatrix} = \begin{pmatrix} 0 \end{pmatrix}$$

y

$$\Lambda((A')B)C = \begin{pmatrix} 0 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 2 \end{pmatrix} = \begin{pmatrix} 2 \end{pmatrix}$$

Este método funciona independientemente del largo del *mensaje codificado*. Intuitivamente, la representación binaria en la matriz B no ocupa estar completa (llegar hasta una potencia de 2), C solo depende del número de columnas de B, la función Λ no depende de nada, y, como se demostró anteriormente, la codificación necesaria para los códigos de Hamming funciona con número distintos a potencias de 2. Adicionalmente, en el repositorio de *GitHub* correspondiente al proyecto se llevan a cabo pruebas con *PyTest* para probar el funcionamiento con distintos largos del *mensaje*.

En conclusión, la solución aquí presentada, que adapta el método de Hamming desde el software al álgebra lineal, se basa en 3 matrices:

- A: Matriz horizontal ($1 \times m, m \in N$) con el mensaje ya codificado. Puede o no puede tener alteraciones.

- B: Matriz rectangular $(m \times n, n \in N)$ donde el reglón k contiene la representación binaria de k en sus columnas y n es el máximo número de bits necesarios para representar la última posición del mensaje, es decir, $\lceil \log_2(m) \rceil$
- C: Matriz vertical $(n \times 1)$ donde el elemento $A_{k1} = k$.

Además de una función Λ que traduce los elementos de una matriz en Z_2 al campo Z.

Al combinar estos elementos de la siguiente manera:

$$\Lambda(AB)C$$

se obtiene la posición donde ocurrió la alteración o 0 de lo contrario. Con esta información, es puede corregir el error si así se desea.

5 Conclusiones

6 Anexos

6.1 Código en Google Colab

6.2 Código en GitHub

6.3 ¿Cómo se acelera la múltiplicación de matrices?

7 Bibliografía