Universidad del Valle De Guatemala

Facultad de Ingeniería

Redes

Laboratorio 2: Esquemas de detección y corrección de errores

Javier Mombiela 20067
Javier Valle 20159

Guatemala, 30 de mayo 2023

Descripción

A lo largo del desarrollo de la presente práctica se desarrollaron dos algoritmos que sirvieron para detectar y corregir errores en tramas de bits. Asimismo, con el desarrollo de los algoritmos se especificó que trama se ingresó, la paridad de los bits, los bits de paridad y el código encriptado. Cabe destacar que estos algoritmos se programaron en dos lenguajes distintos con el objetivo de crear un emisor y un receptor capaces de encriptar y desencriptar las tramas enviadas respectivamente. Es importante mencionar que los algoritmos utilizados en el presente laboratorio fueron: Algoritmo de Hamming y Algoritmo de Checksum.

Algoritmo de Hamming

El algoritmo de Hamming busca agregar bits de paridad a un mensaje original con el fin de que este mensaje sea recibido, propiamente, por un receptor y revisado por este mismo con el fin de determinar si la trama recibida con la información de los bits de paridad fue la misma que se envió desde un principio en el sistema de comunicación. En caso de que se detecte que la paridad no coincide o que haya algún error en el contenido de la trama, entonces se reportaría un error al usuario que recibió el mensaje entramado.

Asimismo, es importante mencionar que este algoritmo busca como tarea agregar bits de paridad en sus posiciones correspondientes para poder, de alguna manera, “encriptar” los mensajes enviados por parte de un emisor. También es importante mencionar que este algoritmo puede funcionar a la perfección para corregir errores en tramas defectuosas, cambiando así los bits en los que se había detectado el error.

Algoritmo CheckSum

El algoritmo de CheckSum es un algoritmo de detección de errores. El objetivo de este algoritmo es poder identificar si el mensaje transmitido fue alterado durante la transmisión o el almacenamiento. Este algoritmo consiste en dividir el mensaje en bloques de 8,16 o 32 bits, en nuestro caso se eligió 8 como tamaño del bloque. Luego se suman todos los bloques, lo que resulta en un último bloque de la misma cantidad de bits. Cabe mencionar que la suma se hace utilizando el complemento de 1. A la suma, también se le aplica el complemente, es decir, se cambian todos los bits al valor opuesto y este último bloque es conocido como el CheckSum. El checksum es agregado al mensaje original y se envía al receptor.

Del lado del receptor, también es algo muy sencillo, el tiene que dividir el mensaje en el mismo tamaño de bloques, si es necesario y deberá de sumar todos estos bloques, incluyendo el checksum que recibió. Si la suma de estos bloques es igual a 0, significa que no hay errores y de lo contrario si hay errores, y la trama se descarta.

Resultados

Hamming sin errores

Para Hamming se utilizaron los siguientes códigos: (12,8), (21,16) y (38,32). Se puede observar que si no hay errores, el código retorna un mensaje de que no hay errores y regresa la misma trama que la original, ya que no hay cambios.

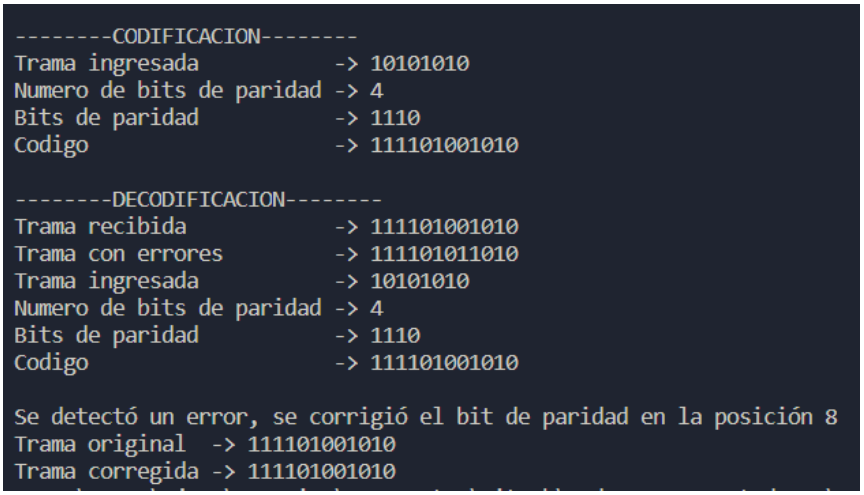
```
--------CODIFICACION--------
Trama ingresada           -> 10101010
Numero de bits de paridad -> 4
Bits de paridad           -> 1110
Codigo                    -> 111101001010

--------DECODIFICACION--------
Trama recibida            -> 111101001010
Trama con errores         -> 111101001010
Trama ingresada           -> 10101010
Numero de bits de paridad -> 4
Bits de paridad           -> 1110
Codigo                    -> 111101001010

No se detectaron errores.
Trama original  -> 111101001010
Trama corregida -> 111101001010
```

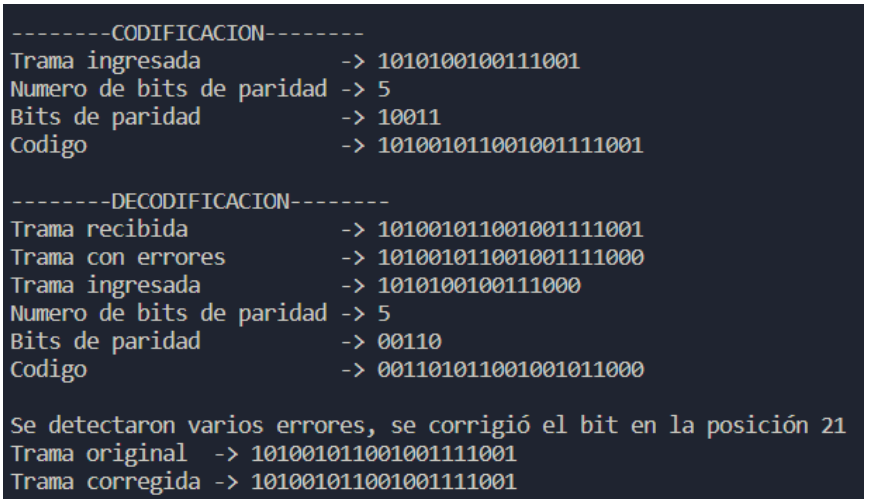
```
--------CODIFICACION--------
Trama ingresada           -> 1010100100111001
Numero de bits de paridad -> 5
Bits de paridad           -> 10011
Codigo                    -> 101001011001001111001

--------DECODIFICACION--------
Trama recibida            -> 101001011001001111001
Trama con errores         -> 101001011001001111001
Trama ingresada           -> 1010100100111001
Numero de bits de paridad -> 5
Bits de paridad           -> 10011
Codigo                    -> 101001011001001111001

No se detectaron errores.
Trama original  -> 101001011001001111001
Trama corregida -> 101001011001001111001
```

```
--------CODIFICACION--------
Trama ingresada           -> 10000100001001001110001010011001
Numero de bits de paridad -> 6
Bits de paridad           -> 100101
Codigo                    -> 10100001010000100010011100010101011001

--------DECODIFICACION--------
Trama recibida            -> 10100001010000100010011100010101011001
Trama con errores         -> 10100001010000100010011100010101011001
Trama ingresada           -> 10000100001001001110001010011001
Numero de bits de paridad -> 6
Bits de paridad           -> 100101
Codigo                    -> 10100001010000100010011100010101011001

No se detectaron errores.
Trama original  -> 10100001010000100010011100010101011001
Trama corregida -> 10100001010000100010011100010101011001
```

Hamming con 1 error

Para poder observar como funciona el código a la hora de cambiar un bit manualmente, se cambió un bit de paridad en la posición 8, en la primera trama. Se puede observar que el código logra identificar que solo hubo un error en las paridades y sabe que es un bit de paridad el incorrecto. Para las otras dos tramas se cambió un bit manualmente en las posiciones 21 y 17, respectivamente, y se puede observar que el código logra corregir dichos errores.





Hamming con 2 errores

Para poder ver como se comporta el código a la hora de cambiar 2 bits manualmente en el receptor, se utilizaron tramas de 32 bits, y a la hora de hacer 2 cambios, se puede observar que el código detecta que hay más de un error, pero no logra detectar las posiciones ni corregirlos.

```
--------CODIFICACION--------
Trama ingresada           -> 10000100001001001110001010011001
Numero de bits de paridad -> 6
Bits de paridad           -> 100101
Codigo                    -> 10100001010000100010011100010101011001

--------DECODIFICACION--------
Trama recibida            -> 10100001010000100010011100010101011001
Trama con errores         -> 10100001010000100010011100010101011111
Trama ingresada           -> 10000100001001001110001010011111
Numero de bits de paridad -> 6
Bits de paridad           -> 000101
Codigo                    -> 00100001010000100010011100010101011111

Se ha detectado mas de un error.
```

```
--------CODIFICACION--------
Trama ingresada           -> 10000111111001001110001010011001
Numero de bits de paridad -> 6
Bits de paridad           -> 101101
Codigo                    -> 10110001011111100010011100010101011001

--------DECODIFICACION--------
Trama recibida            -> 10110001011111100010011100010101011001
Trama con errores         -> 01110001011111100010011100010101011001
Trama ingresada           -> 10000111111001001110001010011001
Numero de bits de paridad -> 6
Bits de paridad           -> 101101
Codigo                    -> 10110001011111100010011100010101011001

Se ha detectado mas de un error.
```

```
--------CODIFICACION--------
Trama ingresada           -> 10000111111001001110001010011010
Numero de bits de paridad -> 6
Bits de paridad           -> 011101
Codigo                    -> 01110001011111100010011100010101011010

--------DECODIFICACION--------
Trama recibida            -> 01110001011111100010011100010101011010
Trama con errores         -> 10110001011111100010011100010101011010
Trama ingresada           -> 10000111111001001110001010011010
Numero de bits de paridad -> 6
Bits de paridad           -> 011101
Codigo                    -> 01110001011111100010011100010101011010

Se ha detectado mas de un error.
```

CheckSum sin errores

A la hora de utilizar el algoritmo de CheckSum con las mismas tramas, podemos observar que se utilizaron tramas de 8,16 y 32 bits respectivamente. Se puede observar que ya que no hay errores, el código retorna un mensaje de que la trama recibida por el receptor no contiene errores.

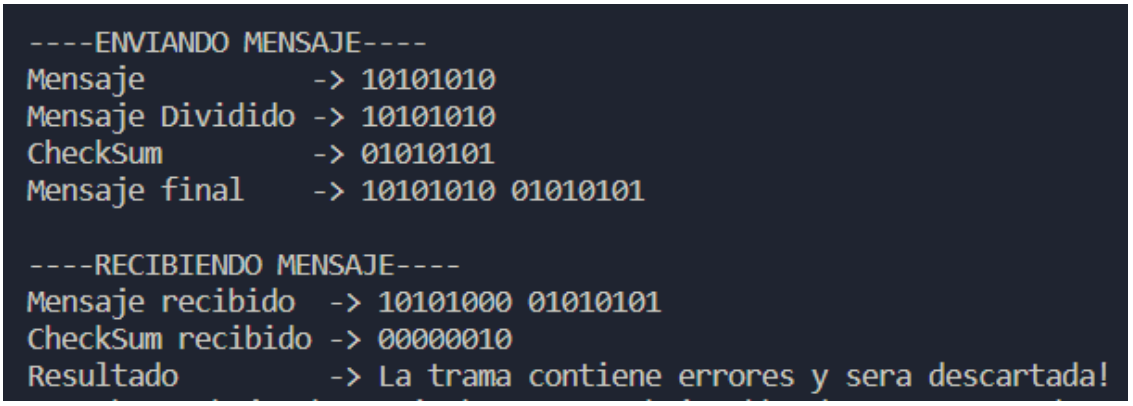
```
----ENVIANDO MENSAJE----
Mensaje          -> 10101010
Mensaje Dividido -> 10101010
CheckSum         -> 01010101
Mensaje final    -> 10101010 01010101

----RECIBIENDO MENSAJE----
Mensaje recibido  -> 10101010 01010101
CheckSum recibido -> 00000000
Resultado         -> La trama no contiene errores!
```

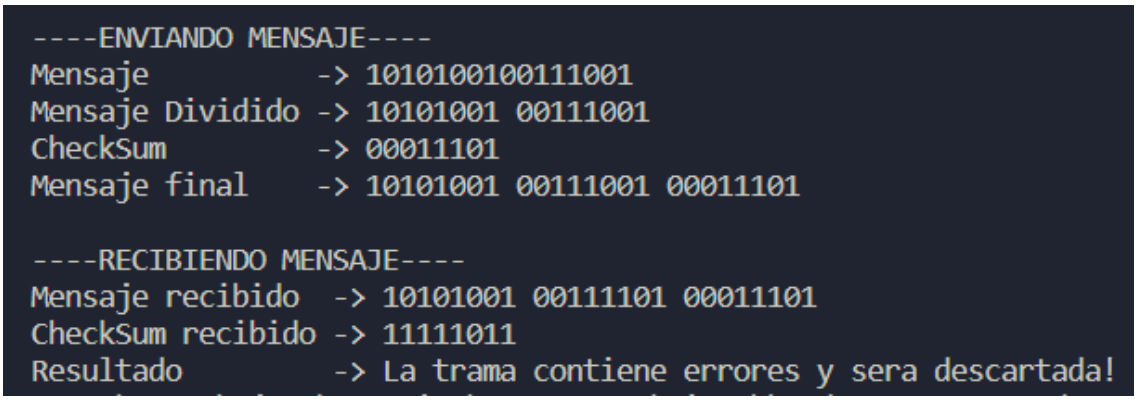
```
----ENVIANDO MENSAJE----
Mensaje          -> 1010100100111001
Mensaje Dividido -> 10101001 00111001
CheckSum         -> 00011101
Mensaje final    -> 10101001 00111001 00011101

----RECIBIENDO MENSAJE----
Mensaje recibido  -> 10101001 00111001 00011101
CheckSum recibido -> 00000000
Resultado         -> La trama no contiene errores!
```

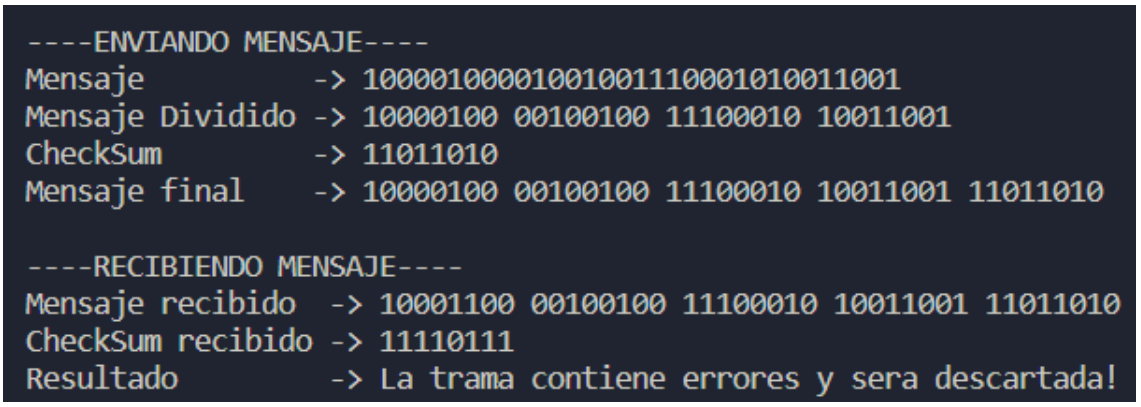
```
----ENVIANDO MENSAJE----
Mensaje          -> 10000100001001001110001010011001
Mensaje Dividido -> 10000100 00100100 11100010 10011001
CheckSum         -> 11011010
Mensaje final    -> 10000100 00100100 11100010 10011001 11011010

----RECIBIENDO MENSAJE----
Mensaje recibido  -> 10000100 00100100 11100010 10011001 11011010
CheckSum recibido -> 00000000
Resultado         -> La trama no contiene errores!
```

CheckSum con 1 error

Para poder ver como se comporta el código a la hora de cambiar un bit manualmente, podemos observar que el código si identifica que hay errores en la trama recibida, pero no sabe en que posición se encuentra.

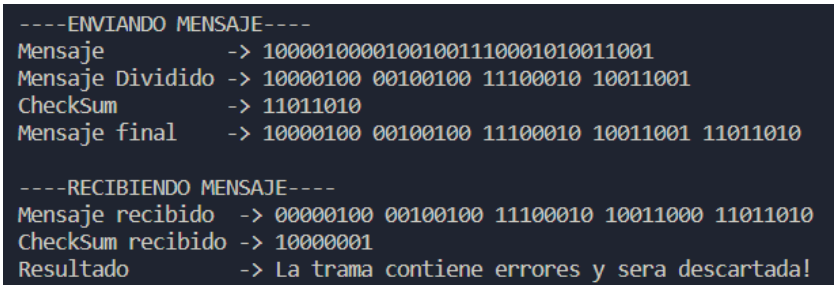


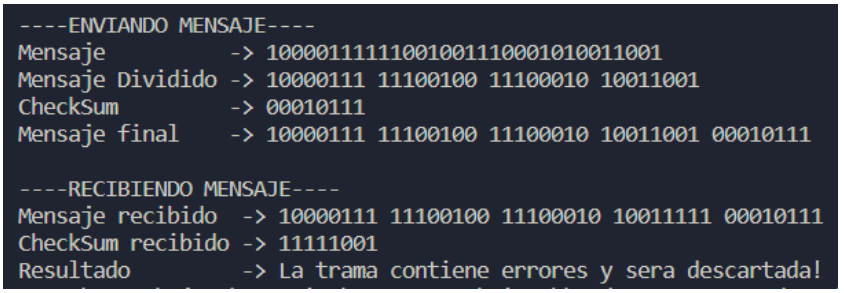


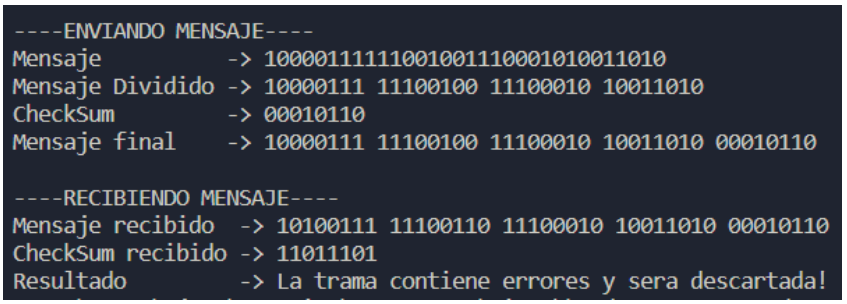


CheckSum con 2 errores

A la hora de cambiar dos bits manualmente en el algoritmo de CheckSum, podemos observar que el resultado es el mismo que cuando se cambió solamente un bit, solo detecta que la trama recibida contiene errores, pero no sabe en qué posición se encuentran.





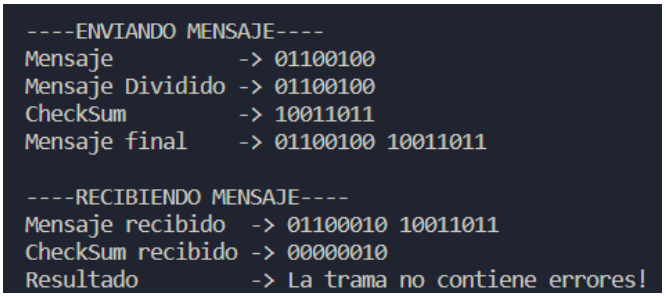


Hamming con errores sin detectar

No es posible encontrar una trama específica con 1 solo error, en donde el código de Hamming no lo detecte.

CheckSum con errores sin detectar

En este caso se ingresó una trama en donde en el primer bloque se cambiaron los bits de la posición 6 y 7, y aun asi, el código no detecta los errores.



Discusión

Al analizar los resultados que se obtuvieron a la hora de realizar el laboratorio, podemos darnos cuenta de algunas cosas. A la hora de utilizar el algoritmo de Hamming podemos observar que el algoritmo es capaz de funcionar bien a la hora de que la trama recibida por el receptor tiene un error o menos. En caso de que la trama no tenga errores, se puede observar que el código retorna un mensaje diciendo que no hay errores, lo cual es correcto, ya que el receptor si está recibiendo la trama correctamente. A la hora de cambiar un bit manualmente, podemos observar que muestra un mensaje diciendo que hay error, en que posición está, y corrige la trama. Este comportamiento también es correcto, ya que el propósito de Hamming es poder detectar y corregir errores. A la hora de tener 2 errores, se puede observar que el algoritmo, falla ya que no logra determinar en las posiciones en las que están los errores, y tampoco logra corregirlos. Esto sucede, debido a que este algoritmo está diseñado para poder detectar y corregir un solo error, esto es debido a los bits de paridad, como es que se calculan y se posicionan, los cuales ayudan a detectar los errores y donde están ubicados. A la hora de intentar ingresar una trama con un solo error en el algoritmo de Hamming, sin que este lo detecte, esto no fue posible. Esto es debido a que los bits de paridad en el algoritmo se colocan en posiciones estratégicas, por lo tanto, si se modifica un bit, todos los demás bits dependen de este, por lo cual cualquier trama con un solo bit modificado, será detectada como un error.

A la hora de utilizar el algoritmo de CheckSum, podemos observar un comportamiento similar al de Hamming a la hora que no hay errores en la trama recibida por el receptor. En este caso, el algoritmo retorna un mensaje de que no hay errores en la trama. A la hora de cambiar un bit manualmente, podemos observar que el código sí logra identificar que hay un error en la trama, pero no sabe en qué posición se encuentra y tampoco lo corrige. Esto también sucede cuando cambiamos 2 bits en la trama recibida por el receptor. Este comportamiento también es el correcto, ya que el algoritmo CheckSum está diseñado solamente para poder identificar errores en la trama y no es un algoritmo que sea capaz de corregir dichos errores. Por lo tanto, el algoritmo sí es capaz de detectar los cambios en los datos, pero no nos proporciona información sobre la ubicación exacta de los errores. Esto se debe a que la suma de verificación se calcula agregando bits de todos los bloques, lo que dificulta la identificación de la posición específica de los bits que presentan errores. A la hora de ingresar una trama específica con errores que no logre detectar, se ingresó la siguiente trama: “01100100” y se modificó a este valor: “01100010”. Se puede ver que se cambiaron dos bits manualmente, por lo que debería de haber errores, pero el código no logro identificarlos. Esto es debido a que el algoritmo CheckSum tiene sus limitaciones, una de ellas siendo que no logra identificar los errores de transposición. Es decir, si dos bits adyacentes se intercambian, el checksum no lo detectará porque la suma total de los datos no cambiará. Por lo tanto, en este caso se modificaron los bits adyacentes de “01” a “10”, causando un error de transposición.

Comentario

El presente laboratorio nos sirvió para poder entender como es que funciona el entramado y encriptado de mensajes a través de redes. Por otro lado, nos fue de bastante utilidad aprender a codificar los algoritmos de Hamming y CheckSum, dado que estos algoritmos nos podrían ser de bastante utilidad para el momento de querer encriptar cualquier mensaje confidencial o mensaje normal que desee compartir a alguien por cualquier medio vulnerable. Con la programación de estos algoritmos, logramos entender como detectar cuando uno de estos es de detección de errores o de detección de errores y corrección de los mismos. Lo anterior fue de bastante utilidad, dado que en muchas ocasiones se está trabajando en un contexto muy estricto con el manejo del encriptado/codificación de tramas y en otras ocasiones el contexto/ambiente no es tan estricto y la pérdida de información no es tan relevante. Finalmente, también logramos entender las limitaciones que llegan a presentar cada uno de estos algoritmos al momento de trabajar con las tramas que se les envía a cada uno de estos.

Conclusiones

- Si se desea detectar y corregir más de un error dentro de una trama de bits, se recomienda usar otro algoritmo que no sea el de Hamming, ya que este algoritmo es eficiente para detectar y corregir errores cuando se trata de un solo error en la trama.
- El algoritmo de CheckSum es muy eficiente para detectar errores, pero no es muy eficiente a la hora de darnos más información, ya que este solo logra detectar si hay errores o no, pero no logra identificar el error, ni la cantidad de errores que hay.
- Otra limitación de CheckSum es que este algoritmo no detecta errores de transposición, lo que lo hace menos efectivo, ya que pueden haber cambios específicos en la trama y el algoritmo no lo detectara.
- Los algoritmos usados son bastante útiles dependiendo del contexto en el que se desean usar. Por ejemplo, si se desea detectar el error, pero no corregirlo, entonces se recomienda usar CheckSum. Sin embargo, si se desea detectar y corregir errores, entonces se recomienda usar el algoritmo de Hamming.

Bibliografía

itskawal2000. (16 de julio del 2021). Error Detection Code - CheckSum. Recuperado de: https://www.geeksforgeeks.org/error-detection-code-checksum/

Pandey, H. (18 de mayo del 2023). Hamming Code in Computer Network. Recuperado de: https://www.geeksforgeeks.org/hamming-code-in-computer-network/

Tanenbaum y Wetherhall. Quinta Edición. Página 243.