

Universidad del Valle de Guatemala
CC3067 - Redes
Adrian Rodríguez 21691
Samuel Chamalé 21881

Laboratorio #2 - Esquemas de detección y corrección de errores

Primera parte

Repositorio

<https://github.com/chamale-rac/detection-correction/tree/main/receiver>

Lenguajes utilizados

- Rust: Emitter
- Go: Receptor

Mensajes

- 0100100001000101010011000100110001001111 = “hello” en ascii
- 01101001011011100110110101100001011000110111010101101100011000010110010
001101111 = “inmaculado” en ASCII
- 01100001011001000110100101101111011100110010000001100001001000000111010
001101111101100100011011110111001100100000011011010110100101110011001000
00011000010110110101101001011001110110111101110011 = “adios a todos mis
amigos” en ASCII

3 Mensajes sin cambio

Primer Algoritmo (Hamming 7,4)

Primer Mensaje

```
~/detection-correction/sender/src main 06:17:23 PM
) cargo run hamming
Finished dev profile [unoptimized + debuginfo] target(s) in 0.05s
Running /Users/adrian/detection-correction/sender/target/debug/sender ham
ming
>> (n): 7
>> (m): 4
The number of redundancy bits is sufficient.
Enter the binary message: 01001000010001010100100110001001111
The hamming code is: 100110011100001001100010010110011000111001001100010
0110011111111

~/detection-correction/sender/src main 10s 06:25:59 PM
) []

~/detection-correction/receiver main 06:29:00 PM
) go run main.go hamming
>> (n): 7
>> (m): 4
Enter the encoded binary message: 10011001110000100110001001011001100011110010
11000111100100110011111111
The final decoded message is: 01001000010001010100100110001001111

~/detection-correction/receiver main 6s 06:29:07 PM
) python
Python 3.10.14 (main, Jul 17 2024, 16:52:01) [Clang 15.0.0 (clang-1500.3.9.4)]
on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> "010010000100010101001000100110001001111" == "01001000010001010100100010
110001001111"
True
```

Segundo Mensaje

```

~/detection-correction/sender/src main 06:32:54 PM
) cargo run hamming
    Finished `dev` profile [unoptimized + debuginfo] target(s) in 0.00s
    Running `~/Users/adrian/detection-correction/sender/target/debug/sender hamming`
>> (n): 7
>> (m): 4
The number of redundancy bits is sufficient.
Enter the binary message: 01101001011011100110110101100001011000110111010110110001100010110001101111
The hamming code is: 110011000110011100110001011011001101010101100110110100111
001101000011000111101001011100110011110011001101101001110011001100110111111

~/detection-correction/sender/src main 2m 23s 06:35:17 PM
) []

~/detection-correction/receiver main 06:31:10 PM
) go run main.go hamming
>> (n): 7
>> (m): 4
Enter the encoded binary message: 110011000110011100110001011011001101010101110
01101101001110011010000110001111010010111001100111100110011011001110011010011
0011001101111111
The final decoded message is: 01101001011011100110101011000010110001011101010
1101100011000010110010001101111

~/detection-correction/receiver main 4s 06:35:27 PM
) python
Python 3.10.14 (main, Jul 17 2024, 16:52:01) [Clang 15.0.0 (clang-1500.3.9.4)]
on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> "011010010110111001101010110000101100010111010101101100011000010110010001
101111" == "01101001011011100110101011000010110001011101010110110001100001011
0010001101111"
True

```

Tercer Mensaje

```

~/detection-correction/sender/src main 06:36:24 PM
) cargo run hamming
    Finished `dev` profile [unoptimized + debuginfo] target(s) in 0.00s
    Running `~/Users/adrian/detection-correction/sender/target/debug/sender hamming`
>> (n): 7
>> (m): 4
The number of redundancy bits is sufficient.
Enter the binary message: 0110000101100100011010010110111011100110010000001100
00100100000011010001011101100100011011101110011001000000110101011010010111
0011001000000110000101101011010010110011101110111011100111
The hamming code is: 11001101101001110011010011001100110001100111001101111100
01111100001101010100000001100110101001010101000000000111100110011001101111
111110011010011001100110111111000111100001101010100000001100110101010110011
000110010001111100001101010100000001100110110110011100110011000110011
100110000111110011011111100011111000011

~/detection-correction/sender/src main 8s 06:36:41 PM
) []

~/detection-correction/receiver main 06:36:52 PM
) go run main.go hamming
>> (n): 7
>> (m): 4
Enter the encoded binary message: 110011011010011100110100110011001100011001110
0110111111000111110000110101010000000011001101101001010101000000000011110011
0011001101111111110011010011001100110111111000111100001101010100000001100110
10101011001100011001000111110000110101010000000011001101100111001101010111
0011000110011001100001111100110111110001111000011
The final decoded message is: 011000010110010001101001011011101110011001000000
110000100100000011101000110111101100100011011101110011001000000101101011001
011100110010000001100001011011010101010011011011011101110110011

~/detection-correction/receiver main 7s 06:36:59 PM
) python
Python 3.10.14 (main, Jul 17 2024, 16:52:01) [Clang 15.0.0 (clang-1500.3.9.4)]
on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> "0110000101100100011010010110111101110011001000000110000100100000011010001
1011101100100011011101110011001000000110101010100111001100100000011000010
110110101010011001110011011011101110011" == "0110000101100100011010000110101101
11001100100000011000010010000001101000110111011001000110111011001100100000
110110101010010110011001000000110000101101101010100110110111011101110011
"
True

```

Segundo Algoritmo (CRC 32)

Primer Mensaje

```

~/detection-correction/sender/src main 06:42:33 PM
) cargo run crc32
    Finished `dev` profile [unoptimized + debuginfo] target(s) in 0.04s
    Running `~/Users/adrian/detection-correction/sender/target/debug/sender crc32`
Enter the binary message: 0100100001000101010011000100110001001111
Transmitted message: 01001000010001010100110001001111110010111100010111011000100110
11011000100110

~/detection-correction/sender/src main 5s 06:45:02 PM
) []

~/detection-correction/receiver main !2 06:48:53 PM
) go run main.go crc32
Enter the received message: 0100100001000101010011000100111111001011110
00110110111000100110
Frame is correct.
The original message is: 0100100001000101010011000100110001001111

~/detection-correction/receiver main !2 3s 06:48:57 PM
) python3
Python 3.10.14 (main, Jul 17 2024, 16:52:01) [Clang 15.0.0 (clang-1500.3.9.4)]
on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> "0100100001000101010011000100110001001111" == "0100100001000101010011000100
110001001111"
True

```

Segundo Mensaje

```

~/detection-correction/sender/src main 06:51:03 PM
) cargo run crc32
    Finished `dev` profile [unoptimized + debuginfo] target(s) in 0.00s
    Running `~/Users/adrian/detection-correction/sender/target/debug/sender crc32`
Enter the binary message: 01101001011011100110110101000010110001101110101101
100011000010110010001101111
Transmitted message: 011010010110111001101101011000010110001101110101011010001
10000101100100011011110100000010100011011000011011000
~/detection-correction/sender/src main 06:51:08 PM
) []

~/detection-correction/receiver main 06:51:12 PM
) go run main.go crc32
Enter the received message: 011010010110111001101101011000010110001101110101011
01100011000010110010001101111010000001010001101100001011000
Frame is correct.
The original message is: 011010010110111001101101011000010110001101110101011011
00011000010110010001101111

~/detection-correction/receiver main 4s 06:51:54 PM
) python3
Python 3.10.14 (main, Jul 17 2024, 16:52:01) [Clang 15.0.0 (clang-1500.3.9.4)]
on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> "011010010110111001101101011000010110001101110101011000110000101100010001
101111" == "0110100101101110011011010110000101100011011101010110110001100001011
0010001101111"
True

```

Tercer Mensaje

```
~/detection-correction/sender/src main 06:52:46 PM ~/detection-correction/receiver main 06:53:31 PM
) cargo run crc32
    Finished `dev` profile [unoptimized + debuginfo] target(s) in 0.00s
    Running `~/Users/adrian/detection-correction/sender/target/debug/sender crc
32'
Enter the binary message: 01100001011001000110100101101111011100110010000001100
0010010000000111010010111101100100010111101110011000000010110010110010111
001100100000001100001011010101101001011001101011101110111011101110110011
0011001000000110000101101010110100101100110101110111011101110111011101110000
011100110001100
Frame is correct.
The original message is: 0110000101100100011010010110111101110111001100000011000
010010000001110100010110110011011100110011000000011010110110100101110
0110010000001100000101101010110100101100110110110111011101110011
~/detection-correction/receiver main 8s 06:53:40 PM
) python
Python 3.10.14 (main, Jul 17 2024, 16:52:01) [Clang 15.0.0 (clang-1500.3.9.4)]
on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> "0110000101100100011010010110111101110011001000001100000011010001
10111011001000101110111011100110000001101011011011100110010000011000010
1101010110100101100110110111101110011"=="01100001011001000110100101110111
10011001000000110000010100000011100010111101100100001011110111001100010000
101101011010010111001100100000011000010110101011000101101110111011101110111
True
```

3 Mensajes con bit cambiado (detecta o corrige)

Primer Algoritmo (Hamming 7,4)

Primer Mensaje (Antepenúltimo bit cambiado): *Corregido correctamente.*

```
chama => sender> cargo run hamming
Finished 'dev' profile [unoptimized + debuginfo] target(s) in 0.02s
Running 'target/debug/sender.exe hamming'
>> (n): 7
>> (m): 4
The number of redundancy bits is sufficient.
Enter the binary message: 010010000100010101001100010011001001111
The hamming code is: 100110011100001001100010010110011110010011100011101001100
1111111
chama => sender>
```

Segundo Mensaje (Segundo bit cambiado): *Corregido correctamente.*

```
chama => sender> cargo run hamming
Finished `dev` profile [unoptimized + debuginfo] target(s) in 0.01s
Running `target/debug/sender.exe hamming`
>> (n): 7
>> (m): 4
The number of redundancy bits is sufficient.
Enter the binary message: 01101001101110110110000101100011011010110110001
100001101001000110111
The hamming code is: 1100110001100111001100010110110011010111011010011100110
1000011000111010011100110011100110011011001100110011011111
chama => sender>

chama => receiver> go run main.go hamming
>> (n): 7
>> (m): 4
Enter the encoded binary message: 1000110001100111001100010110110011010110111001101
101001110011010000110001110110010111001100110011011011001101100110011001101
111111
Error at position 2
The final decoded message is: 011010011011100110110110110000101100011011101011011
000110000111001000110111
chama => receiver>
```

Tercer Mensaje (Quinto bit cambiado): *Corregido correctamente.*

```
chama => sender> cargo run hamming
Finished `dev` profile [unoptimized + debuginfo] target(s) in 0.01s
Running `target/debug/sender.exe hamming`
>> (n): 7
>> (m): 4
The number of redundancy bits is sufficient.
Enter the binary message: 0110000101001000110010110111011001100100000011000100
1000000110100011101101001000011011101100110000001010101010010110011001000
100110000101010101010010110101101110110011
The hamming code is: 11001101100110011001100110011001100110011011111110001111
1000011010100000000101011010101010000000001111011001100110111111100110
100110010010101111110000111100001010101000000001001101010101100110001000111
10000110101010000000010011010101011001101010101100110001000011111100110
11111110001111000011
chama => sender>
```

Segundo Algoritmo (CRC 32)

Primer Mensaje (Antepenúltimo bit cambiado): *Error detectado.*

[illegible]

Segundo Mensaje (Segundo bit cambiado): *Error detectado.*

```
chama => sender> cargo run crc32
Finished 'dev' profile [unoptimized + debuginfo] target(s) in 0.01s
Running 'target/debug/sender.exe crc32'
Enter the binary message: 0110100101101110011011011000010110001101101010110110001
1000010110010001101111
Transmitted message: 011010010110111001101101100001011000101110101011011000110000
10110010001101111010000001010001101100011011000
chama => sender>

chama => receiver> go run main.go crc32
Enter the received message: 0010100101101110011011010110000101100011011010101101100
0110000101100100011011110100000010100011011000011011000
Frame is incorrect.
chama => receiver>
```

Tercer Mensaje (Quinto bit cambiado): *Error detectado.*

```
chama => sender> cargo run crc32
Finished 'dev' profile [unoptimized + debuginfo] target(s) in 0.01s
Running 'target/debug/sender.exe crc32'
Enter the binary message: 011000010110010001101001011011101110011001000000110000100
1000000111010001101111011001100110010000001011010110100101110011001000
000110000101101101010100110011011011101110011
Transmitted message: 0110000101100100011010010110111011100110010000001100010010000
00110100001101110110011001100110010010000001101011010010111001100100100000011
000101101101011010011011011101110110011001111100000011100110001100
chama => sender>

chama => receiver> go run main.go crc32
Enter the received message: 0110100101100100011010010110111011100110010000001100001
0010000001110100011011110110010001101110110011001000000110110101101001011100110010
0000011000010110110101100101100110110111011001101101111100000011100110001100
Frame is incorrect.
chama => receiver>
```

3 Mensajes con 2 o más bits cambiados (detecta o corrige)

Primer Algoritmo (Hamming 7,4)

Primer Mensaje (Primer y último bit cambiado): *Corregido correctamente.*

```
chama => sender> cargo run hamming
Finished 'dev' profile [unoptimized + debuginfo] target(s) in 0.02s
Running 'target/debug/sender.exe hamming'
>> (n): 7
>> (m): 4
The number of redundancy bits is sufficient.
Enter the binary message: 0100100001000101010011000100110001001111
The hamming code is: 10011001110000100110001001011001110010011100100111001001100
11111111
chama => sender>

chama => receiver> go run main.go hamming
>> (n): 7
>> (m): 4
Enter the encoded binary message: 0001100111000010011000100101100110001110010011000
11110010011001111110
Error at position 1
Error at position 70
The final decoded message is: 0100100001000101010011000100110001001111
chama => receiver>
```

Segundo Mensaje (Primeros dos bits cambiados): *No lo pudo detectar bien, pues estaban en el mismo “batch”.*

```
chama => sender> cargo run hamming
Finished 'dev' profile [unoptimized + debuginfo] target(s) in 0.01s
Running 'target/debug/sender.exe hamming'
>> (n): 7
>> (m): 4
The number of redundancy bits is sufficient.
Enter the binary message: 01101001011011100110110110000101100011011101010110110001
1000010110010001101111
The hamming code is: 11001100011001110011000101101100110101011100110110011100110
100001100011110100111100110011101100110011001100110011001101111111
chama => sender>

chama => receiver> go run main.go hamming
>> (n): 7
>> (m): 4
Enter the encoded binary message: 00001100011001110011000101101100110101010111001101
101001110011010000110001110100101110011001100110110100111001100110011001101
111111
Error at position 3
The final decoded message is: 11101001010111001101010110000101100010111010101011
0001100001011001000101111
chama => receiver>
```

Tercer Mensaje (Primer y penúltimo bit cambiado): *Corregido correctamente.*

```
chama => sender> cargo run hamming
Finished 'dev' profile [unoptimized + debuginfo] target(s) in 0.01s
Running 'target/debug/sender.exe hamming'
>> (n): 7
>> (m): 4
The number of redundancy bits is sufficient.
Enter the binary message: 011000010110010001101001011011101110011001000000110000100
1000000111010001101111011001000110111001100100000011011010110100101110011000
00011000010110110101010010100111011011101110011
The hamming code is: 110011011010011100110100110011001100110011001101111110001111
10000110101010000000011001101101010101000000000011110011001100110111111100110
10011011001100111111000111100011010101000000001100110101011100110001100100011111
10000110101010000000110011010101110001111000011010101010100000000110011001100
11111110001111000001
chama => sender>

chama => receiver> go run main.go hamming
>> (n): 7
>> (m): 4
Enter the encoded binary message: 00001100011001110011000101101100110101010111001101
1010011100110100001100011101001011001100110011011001110011001100110011001101
111111
Error at position 3
The final decoded message is: 11101001010111001101010110000101100010111010101011
0001100001011001000101111
chama => receiver> go run main.go hamming
>> (n): 7
>> (m): 4
Enter the encoded binary message: 01001101101001110011010011001100011001100111001101
1111110001111100001101010100000001100110101010101000000000111100110011001101
11111110011010011001100110111110001111000011010101000000001100110101010111001100
01100100011110000110101010000000110011010011100110101010110011000110011001100
00111110010111111100011111000001
Error at position 1
Error at position 335
The final decoded message is: 01100001011001000110100101101111011001100100000011000
010010000001110100011011101100100011011101110011001000000011010101010010111001100
1000000110000101101101010100110011001101110111011011
chama => receiver>
```

Segundo Algoritmo (CRC 32)

Primer Mensaje (Primer y último bit cambiado): *Error detectado.*

```
chama => sender> cargo run crc32
Finished 'dev' profile [unoptimized + debuginfo] target(s) in 0.01s
Running 'target\debug\sender.exe crc32'
Enter the binary message: 0100100001000101010011000100110001001111
Transmitted message: 0100100001000101010011000100111111001011110001110111011
000100110
chama => sender>

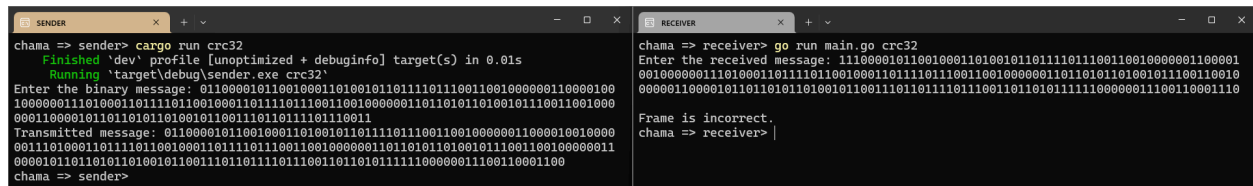
chama => receiver> go run main.go crc32
Enter the received message: 110010000100010101001100010011111100101111000111
011011000100111
Frame is incorrect.
chama => receiver> |
```

Segundo Mensaje (Primeros dos bits cambiados): Error detectado.

```
chama => sender> cargo run crc32
Finished 'dev' profile [unoptimized + debuginfo] target(s) in 0.01s
Running 'target\debug\sender.exe crc32'
Enter the binary message: 01101001011011100110110110000101100011011101010110001
1000010110010001101111
Transmitted message: 0110100101101110011011011000010110001101110101011000110000
1011001000110111110100000010100011011000011011000
chama => sender>

chama => receiver> go run main.go crc32
Enter the received message: 10101001011011100110110101100001011000110111010101100
01100001011001000110111110100000010100011011000011011000
Frame is incorrect.
chama => receiver> |
```

Tercer Mensaje (Primer y penúltimo bit cambiado): *Error detectado.*



```
chama => sender> cargo run crc32
Finished 'dev' profile [unoptimized + debuginfo] target(s) in 0.01s
Running 'target/debug/sender.exe crc32'
Enter the binary message: 0110000101100100011010010110111101110011001000000110000100
10000001101000110111011001000110111011100110010000001101101010100110011001000
00011000010110110110100101100111011011101110011
Transmitted message: 011000010110010001101001011011110111001100100000011000010010000
00111010001101110110010001101111011100110010000001101101011001011100110010000011
0000101101101010110010110011101110111001101101111100000011100110001100
chama => sender>

chama => receiver> go run main.go crc32
Enter the received message: 11100001011001000110100101101111011100110010000001100001
00100000011101000110111101100100011011101110011001000000110110101101001100110010
000001100001011011010110100110011101110111001101101111100000011100110001110
Frame is incorrect.
chama => receiver> |
```

Respuestas

Primer Algoritmo (Hamming 7,4)

¿Es posible manipular los bits de tal forma que el algoritmo seleccionado no sea capaz de detectar el error? ¿Por qué si o por qué no?

Primero, debemos recordar que aunque las pruebas fueron realizadas con Hamming (7,4), los requerimientos indican que debemos ser capaces de manejar cualquier combinación válida (n, k). Nuestro programa está diseñado para soportar esto.

Es importante destacar que no debemos restringirnos a analizar solo el caso (7,4).

Se sabe que los códigos de Hamming pueden detectar errores de uno o dos bits, pero solo pueden corregir un único bit sin detectar errores adicionales.

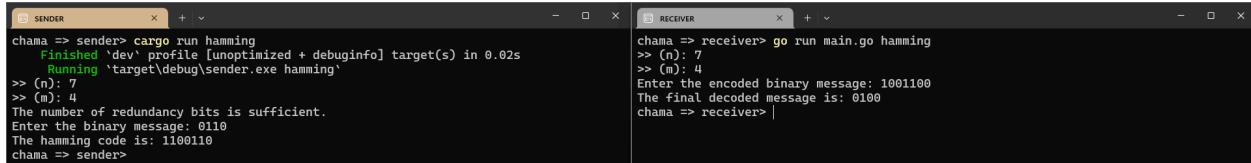
Estas limitaciones son generales para todos los códigos de Hamming y es el caso de tres o más errores el que nos interesa, ya que esto implica que el código de Hamming podría fallar en la detección (y definitivamente no podría corregir estos errores).

En nuestra implementación, el tamaño del mensaje brindado por el usuario se divide en bloques de tamaño k, que luego son transformados a tamaño n durante la codificación (se utiliza un pad de ser necesario). Durante la decodificación, se trabaja en lotes de tamaño n. Esto significa que las limitaciones antes mencionadas no se aplican directamente al mensaje completo que el usuario ingrese, sino a cada bloque individual durante la decodificación.

Esto hace que la posibilidad de no detectar errores disminuya para el mensaje en general, aunque las condiciones se mantengan por lote.

A continuación hemos ejemplificado este worst-case scenario con nuestra implementación:

En el siguiente ejemplo hemos introducido tres errores, distribuidos en las siguiente posiciones: 2, 4 y 6



```
chama => sender> cargo run hamming
Finished `dev` profile [unoptimized + debuginfo] target(s) in 0.02s
Running `target/debug/sender.exe hamming`
>> (n): 7
>> (m): 4
The number of redundancy bits is sufficient.
Enter the binary message: 0110
The hamming code is: 1100110
chama => sender>

chama => receiver> go run main.go hamming
>> (n): 7
>> (m): 4
Enter the encoded binary message: 1001100
The final decoded message is: 0100
chama => receiver> |
```

Vemos que el error no fué detectado.

¿Qué ventajas y desventajas posee este algoritmo con respecto a los otros?

Los código de Hamming ofrecen una ventaja directa sobre Fletcher checksum y CRC-32, ya que estos solo son capaces de detectar errores, mientras que Hamming puede tanto detectar cómo corregir errores.

Encontramos ciertas complejidades al trabajar con lenguajes como Rust, que manejan la memoria de forma estricta. Por ejemplo, para validar la redundancia y utilizar exponentes de base dos, encontramos que números grandes (como 256, 247) podían causar overflow.

En cuanto a la complejidad, haciendo un cálculo sencillo, la codificación tiene una complejidad de al menos $O(n^2)$. Y respecto a la dificultad de implementación, los códigos de Hamming fueron principalmente retadores al intentar comprender la implementación con “syndrome”.

Segundo Algoritmo (CRC 32)

¿Es posible manipular los bits de tal forma que el algoritmo seleccionado no sea capaz de detectar el error? ¿Por qué si o por qué no?

Primero, recordemos que el algoritmo CRC se basa en polinomios. La probabilidad de que un error aleatorio no sea detectado depende del grado del polinomio utilizado, como en el caso del CRC-32. Dada la linealidad y el determinismo del algoritmo, si sustituimos un mensaje completo por otro válido (es decir, una ráfaga completa de errores que forma un nuevo mensaje válido con el mismo checksum), el error no será detectado. Esta es la forma más sencilla de demostrar que el algoritmo puede no detectar un error.

Bajo este concepto, podríamos buscar otro mensaje válido que implique el menor número de errores posible. Sin embargo, esto depende tanto del mensaje enviado como del polinomio utilizado, lo que hace que sea un proceso extremadamente complejo.

En el siguiente ejemplo hemos cambiado la codeword completa, por lo que el error en múltiples bits no será detectado:


```
C:\Program Files\WindowsAp x + -
chama => sender> cargo run crc32
Compiling sender v0.1.0 (C:\Users\chama\Documents\redes\detection-correction\sender)
Finished 'dev' profile [unoptimized + debuginfo] target(s) in 1.61s
Running 'target\debug\sender.exe crc32'
Enter the binary message: 0110
Transmitted message: 011000011010100001100100110110110010
chama => sender> cargo run crc32
Finished 'dev' profile [unoptimized + debuginfo] target(s) in 0.02s
Running 'target\debug\sender.exe crc32'
Enter the binary message: 0111
Transmitted message: 011100011110010001110101000000000101
chama => sender>

C:\Program Files\WindowsAp x + -
chama => receiver> go run main.go crc32
Enter the received message: 011100011110010001110101000000000101
Frame is correct.
The original message is: 0111
chama => receiver>
```

Vemos que el error no fué detectado por CRC 32. Aunque en realidad, para este ejemplo arbitrario, existen 15 errores entre ambos mensajes transmitidos.

```
chama => detection-correction> py .\helpers\binary_errors.py
Enter the first binary message: 011000011010100001100100110110110010
Enter the second binary message: 011100011110010001110101000000000101
Binary Message 1: 011[0]00011[0]10[1][0]00011[0]010[0][1][1]0[1][1]0[1][1]0[0][1][0]
Binary Message 2: 011[1]00011[1]10[0][1]00011[1]010[1][0][0]0[0][0]0[0][0]0[1][0][1]
Number of errors: 15
Error positions: [3, 9, 12, 13, 19, 23, 24, 25, 27, 28, 30, 31, 33, 34, 35]
chama => detection-correction>
```

¿Qué ventajas y desventajas posee este algoritmo con respecto a los otros?

El algoritmo CRC-32 ofrece simplicidad y un menor overhead. Sin embargo, su incapacidad para corregir errores lo coloca en desventaja frente a Hamming y Viterbi, que sí pueden corregir. Tal como lo vimos con nuestras ejecuciones, evitar la detección de errores es poco probable, sin embargo este puede ser engañado deliberadamente si se manipulan tanto los datos como el checksum. Durante la implementación este algoritmo fué más sencillo de implementar que Hamming. Finalmente terminamos el polinomio:

IEEE 802: $x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x^1 + 1$

Aunque cabe destacar que nuestra implementación funcionaría simplemente cambiando la string que indica el polinomio en formato binario.