

UNIVERSIDAD DEL VALLE DE GUATEMALA

Redes, sección 20



LAB # 2

Diego Linares - 221256
Joaquin Campos – 22155

Guatemala, Julio 2025

Mensajes usados (cadenas o tramas)

001101

10111000111

1100110

Corrección

Para correccion de errores se decidio implementar Hamming

Para esta parte se implementaron dos programas: un emisor en Java y un receptor en Python, cumpliendo el requisito de utilizar lenguajes distintos para cada lado.

Emisor (Java)

- Solicita un mensaje binario de 7 bits desde la consola.
- Coloca los bits de datos en las posiciones correspondientes de una trama de 11 bits.
- Calcula los 4 bits de paridad (P1, P2, P4, P8) mediante XOR. El XOR creo que no necesito explicarlo de por qué pero es porque por ejemplo si hubiesen una cantidad par de 1`s pues entonces bit de paridad seria 0, si hubiese una cantidad impar pues bit de paridad 1.
- Imprime la trama codificada completa de 11 bits.

Receptor (Python)

- Solicita una trama binaria de 11 bits desde la consola.
- Convierte la cadena ingresada a una lista de enteros para facilitar el procesamiento.
- Calcula el síndrome sumando las paridades y determinando si existe error y en qué posición.
- Si el síndrome es distinto de 0, corrige el bit correspondiente y muestra la trama corregida y los datos originales.

Funcionamiento del algoritmo

El código Hamming (11,7) consiste en:

- Tomar 7 bits de datos originales y agregar 4 bits de paridad.

- Colocar los bits de paridad en las posiciones potencias de 2 (1, 2, 4, 8) y los datos en las demás.
- Calcular cada bit de paridad para cubrir distintas combinaciones de posiciones.
- En el receptor, recalculamos las paridades sobre la trama recibida para obtener el síndrome, un número que indica exactamente la posición del bit erróneo (si hay uno).
- Si el síndrome es 0, la trama está correcta; si no, se corrige el bit en esa posición.

Prueba realizada primero en el de Java.

Posición	1	2	3	4	5	6	7	8	9	10	11
Tipo	P1	P2	D1	P4	D2	D3	D4	P8	D5	D6	D7

P = bit de paridad

D = bit de datos

Mensaje original:

1 1 0 0 1 1 0

Lo colocamos en las posiciones D1, D2, ..., D7:

Posición	1	2	3	4	5	6	7	8	9	10	11
Valor	P	P	1	P	1	0	0	P	1	1	0

Calcular paridades

Las fórmulas que usa el código Java con lo de los XOR que había mencionado yo anteriormente.

P1 (posición 1): XOR de posiciones 3,5,7,9,11

$$P1 = D1 \wedge D2 \wedge D4 \wedge D5 \wedge D7$$

$$= 1 \wedge 1 \wedge 0 \wedge 1 \wedge 0$$

$$= 1$$

Y así.... Una vez que terminamos de calcular todas las posiciones de de las posiciones de paridad pues

Con los bits de paridad calculados (P1=1, P2=0, P4=1, P8=0) la trama queda así:

Posición	1	2	3	4	5	6	7	8	9	10	11
Valor	1	0	1	1	1	0	0	0	1	1	0

Ahora para python...

Tomamos la convertida y le cambiamos el bit de la posición 11 para ver si lo identifica.

1	2	3	4	5	6	7	8	9	10	11
1	0	1	1	1	0	0	0	1	1	1

Este receptor calcula las paridades y pues devuelve Síndrome: 1011 (binario) = 11 (decimal)

El receptor hace un flip al bit 11: $1 \rightarrow 0$. esto lo hace con otro xor al 1.

La trama corregida vuelve a ser la que envió el emisor:

10111000110

Y ya de ahí pues extraemos las posiciones de los datos

Posición n	3	5	6	7	9	10	11
Valor	1	1	0	0	1	1	0

1100110

Y coincide con el mensaje original.

Versión 2 de corrección de errores:

Para extender la funcionalidad del código Hamming (11,7) y cumplir con el requisito de aceptar cadenas de cualquier longitud, se modificó el emisor para que dividiera el mensaje binario ingresado en bloques de 7 bits. Si el mensaje ingresado tenía una longitud menor a 7, se rellenaba con ceros a la derecha; si era mayor, se dividía en múltiples bloques de 7 bits y se rellenaba el último si era necesario. Cada bloque se codifica por separado utilizando Hamming (11,7), generando una trama de 11 bits por bloque. En el receptor, se incorporó un mecanismo para procesar múltiples

bloques codificados. Se solicita al usuario cuántos bloques desea ingresar y se analiza cada uno de manera individual, detectando y corrigiendo errores si es necesario. Luego, se extraen los 7 bits de datos originales de cada bloque corregido y se reconstruye el mensaje completo concatenando todos los datos. Esta adaptación permite que el sistema funcione con mensajes de cualquier longitud, sin perder la capacidad de detección y corrección por bloque, lo cual se refleja claramente en los resultados mostrados bloque a bloque, seguidos del mensaje completo reconstruido.

Entonces aaaa ya ahora podríamos aceptar cadenas más largas o cortas para la prueba así que vamos con...

10111000111

El emisor.

```
Ingresa un mensaje binario de cualquier longitud: 10111000111

Bloques codificados (11 bits por bloque):
Bloque original: 1011100 ? Codificado: 11100111100
Bloque original: 0111000 ? Codificado: 00011110000
PS C:\Users\diego\Documents\UVG\Septimo Semestre\Redes\Deteccion y correccion errores>
```

El receptor:

```
PS C:\Users\diego\Documents\UVG\Septimo Semestre\Redes\Deteccion y correccion errores> & C:/Users/diego/AppData/Local/Programs/Python/Python313/python.exe "C:/Users/diego/Documents/UVG/Septimo Semestre/Redes/Deteccion y correccion errores/ReceptorHamming.py"
¿Cuántos bloques de 11 bits desea ingresar? 2

Bloque 1:
Ingresa el mensaje recibido (11 bits): 11100111100
No se detectaron errores.
Datos originales: 1011100

Bloque 2:
Ingresa el mensaje recibido (11 bits): 00011110000
No se detectaron errores.
Datos originales: 0111000

Mensaje reconstruido completo:
10111000111000
PS C:\Users\diego\Documents\UVG\Septimo Semestre\Redes\Deteccion y correccion errores>
```

Ahora con un error:

```
PS C:\Users\diego\Documents\UVG\Septimo Semestre\Redes\Deteccion y correccion errores> & C:/Users/diego/AppData/Local/Programs/Python/Python313/python.exe "C:/Users/diego/Documents/UVG/Septimo Semestre/Redes/Deteccion y correccion errores/ReceptorHamming.py"
¿Cuántos bloques de 11 bits desea ingresar? 2

Bloque 1:
Ingresa el mensaje recibido (11 bits): 11100111101
Error detectado en la posición 11. Corrigiendo...
Trama corregida: 11100111100
Datos originales: 1011100

Bloque 2:
Ingresa el mensaje recibido (11 bits): 00011110001
Error detectado en la posición 11. Corrigiendo...
Trama corregida: 00011110000
Datos originales: 0111000

Mensaje reconstruido completo:
10111000111000
PS C:\Users\diego\Documents\UVG\Septimo Semestre\Redes\Deteccion y correccion errores>
```

Notar que cambie solo el ultimo bit a cada cadena y pues funciono, lo detecta y lo corrige.

Ahora dos errores y vamos a probar una cadena de longitud menor a 7

001101

Emisor:

```
Bloque original: 0111000 / Codificado: 0001110000
PS C:\Users\diego\Documents\UAG\Septimo Semestre\Redes\Deteccion y correccion errores> cd "c:\Users\diego\Documents\UAG\Septimo Semestre\Redes\Deteccion y correccion errores" & "C:\Program Files\Java\jdk-24\bin\java.exe" --module-path "D:\ShowCodeDetails\IOExceptionMessages" --cp "C:\Users\diego\AppData\Local\Programs\Python\Python311\python.exe" "C:\Users\diego\Documents\UAG\Septimo Semestre\Redes\Deteccion y correccion errores\EmisorHamming.py"
Ingrese un mensaje binario de cualquier longitud: 001101

Bloques codificados (11 bits por bloque):
Bloque original: 0011010 / Codificado: 11000111010
PS C:\Users\diego\Documents\UAG\Septimo Semestre\Redes\Deteccion y correccion errores>
```

Receptor:

```
PS C:\Users\diego\Documents\UAG\Septimo Semestre\Redes\Deteccion y correccion errores> & C:\Users\diego\AppData\Local\Programs\Python\Python311\python.exe "C:\Users\diego\Documents\UAG\Septimo Semestre\Redes\Deteccion y correccion errores\ReceptorHamming.py"
¿Cuántos bloques de 11 bits desea ingresar? 1

Bloque 1:
Ingrese el mensaje recibido (11 bits): 00000111010
Error detectado en la posición 3. Corrigiendo...
Trama corregida: 00100111010
Datos originales: 1011010

Mensaje reconstruido completo:
1011010
```

Notar que acá ya no es capaz de detectar dos errores porque Hamming no está hecho para ello.

Pruebas de detección:

- Sin errores:

```
=== Emisor Fletcher Checksum ===
Ingrese la trama binaria a enviar: 10111000110
Error: la trama solo puede contener 0s y 1s.

Ingrese la trama binaria a enviar: 10111000110
Tamaño del bloque (8, 16 o 32): 16

Trama original (con padding si aplica): 1011100011000000
Checksum binario: 00000000111100100000000001111001
Trama a enviar: 1011100011000000000000001111001000000001111001

¿Desea verificar otra trama? (s/n): s

Ingrese la trama recibida (solo 0s y 1s): 1011100011000000000000001111001000000001111001
Tamaño del bloque (8, 16 o 32): 16

Checksum recibido: 0000000011110010000000001111001
Checksum calculado: 0000000011110010000000001111001
Trama recibida correctamente: 1011100011000000
```

```
¿Desea enviar otra trama? (s/n): s
Ingrese la trama binaria a enviar: 10111000111
Tamaño del bloque (8, 16 o 32): 8

Trama original (con padding si aplica): 1011100011100000
Checksum binario: 1001100101010010
Trama a enviar: 10111000111000001001100101010010

Ingrese la trama recibida (solo 0s y 1s): 10111000111000001001100101010010
Tamaño del bloque (8, 16 o 32): 8

Checksum recibido: 1001100101010010
Checksum calculado: 1001100101010010
Trama recibida correctamente: 1011100011100000
```

```
=== Emisor Fletcher Checksum ===
Ingrese la trama binaria a enviar: 1100110
Tamaño del bloque (8, 16 o 32): 8

Trama original (con padding si aplica): 11001100
Checksum binario: 1100110011001100
Trama a enviar: 110011001100110011001100

Ingrese la trama recibida (solo 0s y 1s): 11001100110011001100
Tamaño del bloque (8, 16 o 32): 8

Checksum recibido: 1100110011001100
Checksum calculado: 1100110011001100
Trama recibida correctamente: 11001100
```

- Un error:

Desea enviar otra trama? (s/n): s Ingrese la trama binaria a enviar: 1100110 Tamaño del bloque (8, 16 o 32): 8 Trama original (con padding si aplica): 11001100 Checksum binario: 1100110011001100 Trama a enviar: 11001100110011001100	Ingrese la trama recibida (solo 0s y 1s): 110011001100110011000 Tamaño del bloque (8, 16 o 32): 8 Checksum recibido: 1100110011001000 Checksum calculado: 1100110011001100 Error detectado. Trama descartada.
--	---

Error: la trama solo puede contener 0s y 1s. Ingrese la trama binaria a enviar: 001101 Tamaño del bloque (8, 16 o 32): 8 Trama original (con padding si aplica): 00110100 Checksum binario: 0011010000110100 Trama a enviar: 001101000011010000110100	¿Desea verificar otra trama? (s/n): s Ingrese la trama recibida (solo 0s y 1s): 001101000011010000110101 Tamaño del bloque (8, 16 o 32): 8 Checksum recibido: 0011010000110101 Checksum calculado: 0011010000110100 Error detectado. Trama descartada.
--	---

Trama original (con padding si aplica): 00110100 Checksum binario: 0011010000110100 Trama a enviar: 001101000011010000110100 ¿Desea enviar otra trama? (s/n): s Ingrese la trama binaria a enviar: 1100110 Tamaño del bloque (8, 16 o 32): 16 Trama original (con padding si aplica): 1100110000000000 Checksum binario: 00000000110011000000000011001100 Trama a enviar: 110011000000000000000000110011000000000011001100	Checksum calculado: 0011010000110100 Error detectado. Trama descartada. ¿Desea verificar otra trama? (s/n): s Ingrese la trama recibida (solo 0s y 1s): 110011000000000000000000110011000000 000011001101 Tamaño del bloque (8, 16 o 32): 16 Checksum recibido: 00000000110011000000000011001101 Checksum calculado: 00000000110011000000000011001100 Error detectado. Trama descartada.
--	--

- Dos+errores:

Trama a enviar: 110011000000000000000000110011000000000011001100 ¿Desea enviar otra trama? (s/n): 1100110 Finalizando emisor. PS C:\Users\jjcam\Desktop\Semestre_8\Redes\Esquemas-de-deteccion-y-correccion-de-errores\deteccion> python emisor.py === Emisor Fletcher Checksum === Ingrese la trama binaria a enviar: 1100110 Tamaño del bloque (8, 16 o 32): 16 Trama original (con padding si aplica): 1100110000000000 Checksum binario: 00000000110011000000000011001100 Trama a enviar: 110011000000000000000000110011000000000011001100	Checksum calculado: 00000000110011000000000011001100 Error detectado. Trama descartada. ¿Desea verificar otra trama? (s/n): s Ingrese la trama recibida (solo 0s y 1s): 11001100000000000000000011001100 000011001010 Tamaño del bloque (8, 16 o 32): 16 Checksum recibido: 00000000110011000000000011001010 Checksum calculado: 00000000110011000000000011001100 Error detectado. Trama descartada.
---	--

¿Desea enviar otra trama? (s/n): s Ingrese la trama binaria a enviar: 10111000111 Error: la trama solo puede contener 0s y 1s. Ingrese la trama binaria a enviar: 10111000111 Tamaño del bloque (8, 16 o 32): 16 Trama original (con padding si aplica): 1011100011100000 Checksum binario: 00000000100110010000000000110011001 Trama a enviar: 1011100011100000000000001001100100000000110011001	Error detectado. Trama descartada. ¿Desea verificar otra trama? (s/n): s Ingrese la trama recibida (solo 0s y 1s): 101110001110000000000000100110010000 000010011010 Tamaño del bloque (8, 16 o 32): 16 Checksum recibido: 00000000100110010000000010011010 Checksum calculado: 00000000100110010000000010011001 Error detectado. Trama descartada.
--	--

Trama original (con padding si aplica): 1011100011100000 Checksum binario: 00000000100110010000000000110011001 Trama a enviar: 1011100011100000000000001001100100000000110011001 ¿Desea enviar otra trama? (s/n): s Ingrese la trama binaria a enviar: 1100110 Tamaño del bloque (8, 16 o 32): 8 Trama original (con padding si aplica): 11001100 Checksum binario: 1100110011001100 Trama a enviar: 110011001100110011001100	Checksum calculado: 1100110011001100 Trama recibida correctamente: 11001100 ¿Desea verificar otra trama? (s/n): s Ingrese la trama recibida (solo 0s y 1s): 110011001100110011001111 Tamaño del bloque (8, 16 o 32): 8 Checksum recibido: 1100110011001111 Checksum calculado: 1100110011001100 Error detectado. Trama descartada.
---	---

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

Sí, es posible manipular los bits de una trama de tal forma que el algoritmo Fletcher Checksum no detecte el error.

Esto se debe a que Fletcher calcula la validez de una trama a partir de dos sumas acumulativas:

sum1: la suma de todos los bloques de datos.

sum2: la suma acumulada de sum1 a lo largo de la secuencia.

- ¿qué ventajas y desventajas posee cada algoritmo con respecto a los otros dos?

Ventajas del algoritmo Fletcher Checksum

Mejor detección que la paridad o suma simple

Fácil de implementar: Solo requiere sumas y operaciones módulo.

Flexible: Permite usar bloques de 8, 16 o 32 bits según se necesite.

Desventajas del algoritmo

No corrige errores, solo los detecta.

Puede fallar si los errores mantienen las sumas sum1 y sum2

Más overhead que la paridad (agrega dos bloques de checksum).

Sensible al orden de los bloques, lo que complica su análisis en ciertos casos.