Universidad del Valle de Guatemala
18 Avenida 11-95 Guatemala
Facultad de Ingeniería
Departamento de Computación


# Detección y corrección de errores - pt 1

**Integrantes:**
Diego Alexander Hernández Silvestre, 21270
Linda Inés Jiménez Vides, 21169

**Curso:**
Redes

**Sección:**
Sección 10

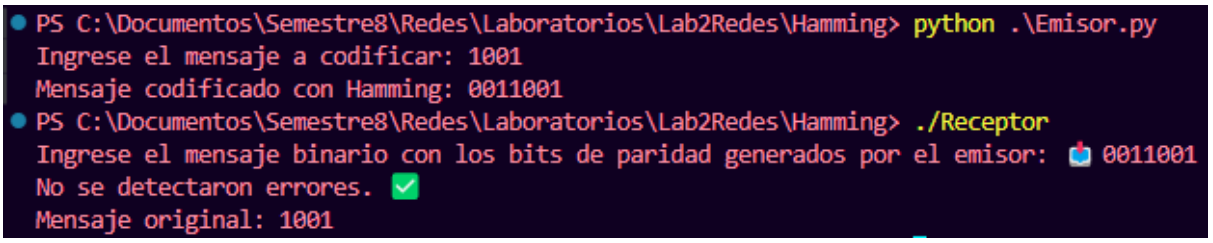
Guatemala, 25 de julio de 2024

**Repositorio GitHub:** https://github.com/Dahernandezsilve/Lab2Redes.git

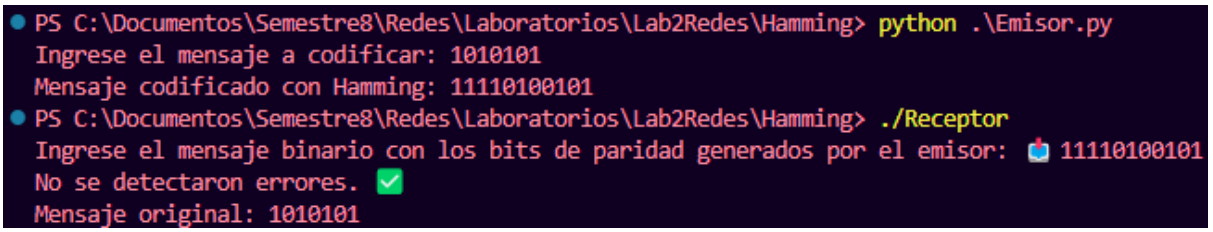
### Escenarios de pruebas

➔ **(sin errores)**: Enviar un mensaje al emisor, copiar el mensaje generado por este y proporcionarlo tal cual al receptor, el cual debe mostrar el mensajes originales (ya que ningún bit sufrió un cambio). Realizar esto para tres mensajes distintos con distinta longitud.

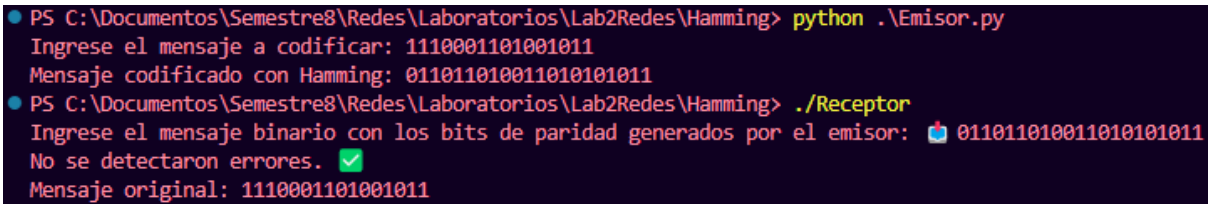
**Algoritmo de corrección de errores: Códigos de Hamming**
**Mensaje 1 - 1001**




**Mensaje 2 - 1010101**




**Mensaje 3 - 1110001101001011**



**Algoritmo de detección de errores: Fletcher checksum**

**Mensaje 1 - 1001**

- **Mensaje codificado - 000010010000010100000010**

```
PS C:\Documentos\Semestre8\Redes\Laboratorios\Lab2Redes\Fletcher_Checksum> python .\Emisor.py
--- Emisor ---
Ingrese el mensaje binario: 1001
Ingrese el tamaño del bloque (8, 16, o 32): 8
Mensaje con checksum: 000010010000010100000010
Checksum en binario: 0000010100000010
PS C:\Documentos\Semestre8\Redes\Laboratorios\Lab2Redes\Fletcher_Checksum> ./Receptor

--- Receptor ---
Ingrese el mensaje binario con el checksum: 000010010000010100000010
Ingrese el tamaño del bloque (8, 16, o 32): 8
No se detectaron errores.
Mensaje original: 00001001
```

**Mensaje 2 - 1010101**

- **Mensaje codificado - 00000000010101010000000000010000000000000000000100**

```
PS C:\Documentos\Semestre8\Redes\Laboratorios\Lab2Redes\Fletcher_Checksum> python .\Emisor.py
--- Emisor ---
Ingrese el mensaje binario: 1010101
Ingrese el tamaño del bloque (8, 16, o 32): 16
Mensaje con checksum: 00000000010101010000000000010000000000000000000100
Checksum en binario: 00000000000100000000000000000100
PS C:\Documentos\Semestre8\Redes\Laboratorios\Lab2Redes\Fletcher_Checksum> ./Receptor

--- Receptor ---
Ingrese el mensaje binario con el checksum: 00000000010101010000000000010000000000000000000100
Ingrese el tamaño del bloque (8, 16, o 32): 16
No se detectaron errores.
Mensaje original: 0000000001010101
```

**Mensaje 3 - 1110001101001011**

- **Mensaje codificado**
  **00000000000000001110001101001011000000000000000000000000000010011100000000000000000000000000001001**

```
PS C:\Documentos\Semestre8\Redes\Laboratorios\Lab2Redes\Fletcher_Checksum> python .\Emisor.py
--- Emisor ---
Ingrese el mensaje binario: 1110001101001011
Ingrese el tamaño del bloque (8, 16, o 32): 32
Mensaje con checksum: 00000000000000001110001101001011000000000000000000000000100111000000000000000000000000001001
Checksum en binario: 00000000000000000000000010011100000000000000000000000000001001
PS C:\Documentos\Semestre8\Redes\Laboratorios\Lab2Redes\Fletcher_Checksum> ./Receptor

--- Receptor ---
Ingrese el mensaje binario con el checksum: 00000000000000001110001101001011000000000000000000000000100111000000000000000000000000001001
Ingrese el tamaño del bloque (8, 16, o 32): 32
No se detectaron errores.
Mensaje original: 00000000000000001110001101001011
```

➔ **(un error)**: Enviar un mensaje al emisor, copiar el mensaje generado por este y cambiar un bit cualquiera antes de proporcionarlo al receptor. Si el algoritmo es de detección debe mostrar que se detectó un error y que se descarta el mensaje. Si el algoritmo es de corrección debe corregir el bit, indicar su posición y mostrar el mensaje original. Realizar esto para tres mensajes distintos con distinta longitud.

**Algoritmo de corrección de errores: Códigos de Hamming**

**Mensaje 1 - 1100**

- **Mensaje codificado - 0111100**
- **Mensaje modificado - 0111101**

```
PS C:\Documentos\Semestre8\Redes\Laboratorios\Lab2Redes\Hamming> python .\Emisor.py
Ingrese el mensaje a codificar: 1100
Mensaje codificado con Hamming: 0111100
PS C:\Documentos\Semestre8\Redes\Laboratorios\Lab2Redes\Hamming> ./Receptor
Ingrese el mensaje binario con los bits de paridad generados por el emisor: 0111101
Se detectaron y corrigieron errores.
Posición del bit a corregir: 7
Mensaje codificado corregido: 0111100
Mensaje corregido: 1100
PS C:\Documentos\Semestre8\Redes\Laboratorios\Lab2Redes\Hamming>
```

**Mensaje 2 - 1110001110**

- **Mensaje codificado - 01101101001110**
- **Mensaje modificado - 01111101001110**

```
PS C:\Documentos\Semestre8\Redes\Laboratorios\Lab2Redes\Hamming> python .\Emisor.py
Ingrese el mensaje a codificar: 1110001110
Mensaje codificado con Hamming: 01101101001110
PS C:\Documentos\Semestre8\Redes\Laboratorios\Lab2Redes\Hamming> ./Receptor
Ingrese el mensaje binario con los bits de paridad generados por el emisor: 01111101001110
Se detectaron y corrigieron errores.
Posición del bit a corregir: 4
Mensaje codificado corregido: 01101101001110
Mensaje corregido: 1110001110
```

**Mensaje 3 - 1001011001101**

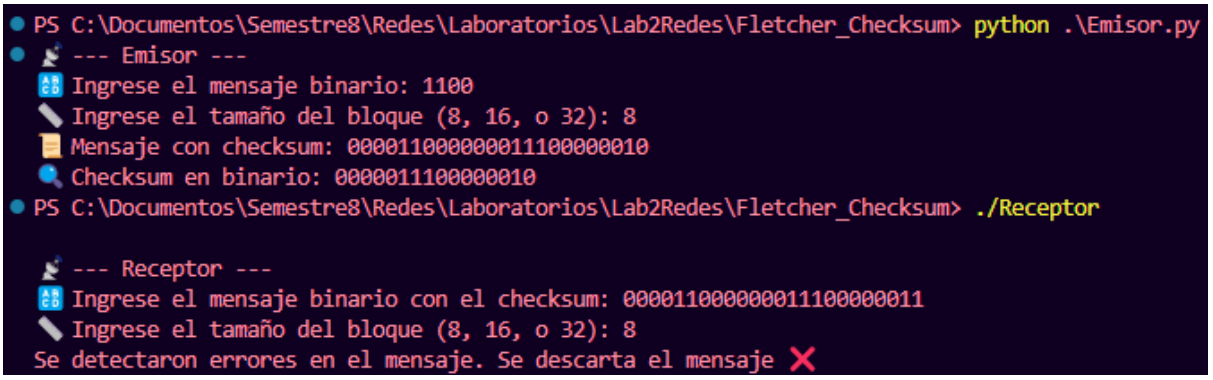
- **Mensaje codificado - 011100100110011101**
- **Mensaje modificado - 111100100110011101**

```
PS C:\Documentos\Semestre8\Redes\Laboratorios\Lab2Redes\Hamming> python .\Emisor.py
Ingrese el mensaje a codificar: 1001011001101
Mensaje codificado con Hamming: 011100100110011101
PS C:\Documentos\Semestre8\Redes\Laboratorios\Lab2Redes\Hamming> ./Receptor
Ingrese el mensaje binario con los bits de paridad generados por el emisor: 111100100110011101
Se detectaron y corrigieron errores.
Posición del bit a corregir: 1
Mensaje codificado corregido: 011100100110011101
Mensaje corregido: 1001011001101
```

## Algoritmo de detección de errores: Fletcher checksum

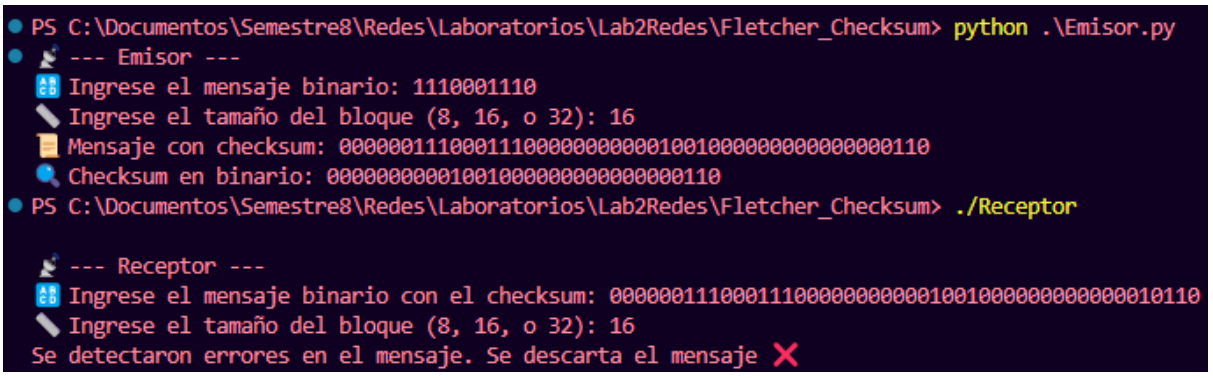
**Mensaje 1 - 1100**

- **Mensaje codificado - 000011000000011100000010**
- **Mensaje modificado - 000011000000011100000011**



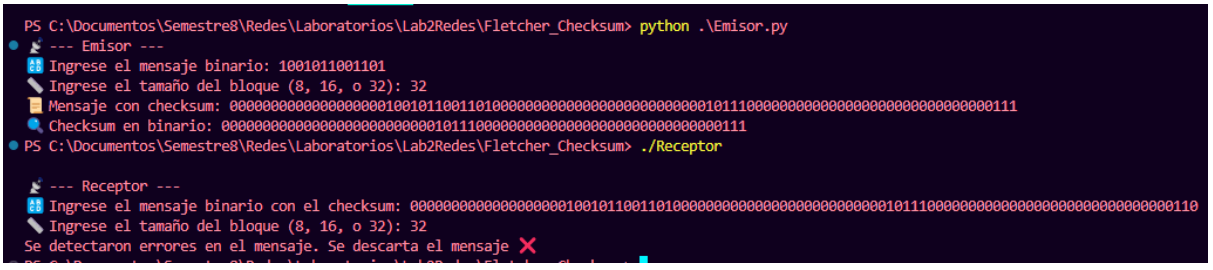

**Mensaje 2 - 1110001110**

- **Mensaje codificado - 00000011100011100000000000100100000000000000000110**
- **Mensaje modificado - 00000011100011100000000000100100000000000000010110**




**Mensaje 3 - 1001011001101**

- **Mensaje codificado**
  **00000000000000000001001011001101000000000000000000000000000001011100000000000000000000000000000111**
- **Mensaje modificado**
  **00000000000000000001001011001101000000000000000000000000000001011100000000000000000000000000000110**



➔ **(dos+ errores):** Enviar un mensaje al emisor, copiar el mensaje generado por este y cambiar dos o más bits cualesquiera antes de proporcionarlo al receptor. Si el algoritmo es de detección debe mostrar que se detectó un error y que se descarta el mensaje. Si el algoritmo es de corrección y puede corregir más de un error, debe corregir los bits, indicar su posición y mostrar el mensaje original. Realizar esto para tres mensajes distintos con distinta longitud.

**Algoritmo de corrección de errores: Códigos de Hamming**

**Mensaje 1 - 1100**

- **Mensaje codificado - 0111100**
- **Mensaje modificado - 0101110**

```
PS C:\Documentos\Semestre8\Redes\Laboratorios\Lab2Redes\Hamming> python .\Emisor.py
Ingrese el mensaje a codificar: 1100
Mensaje codificado con Hamming: 0111100
PS C:\Documentos\Semestre8\Redes\Laboratorios\Lab2Redes\Hamming> ./Receptor
Ingrese el mensaje binario con los bits de paridad generados por el emisor: 0101110
Se detectaron y corrigieron errores.
Posición del bit a corregir: 5
Mensaje codificado corregido: 0101010
Mensaje corregido: 0010
```

**Mensaje 2 - 1110001110**

- **Mensaje codificado - 01101101001110**
- **Mensaje modificado - 01111101001111**

```
PS C:\Documentos\Semestre8\Redes\Laboratorios\Lab2Redes\Hamming> python .\Emisor.py
Ingrese el mensaje a codificar: 1110001110
Mensaje codificado con Hamming: 01101101001110
PS C:\Documentos\Semestre8\Redes\Laboratorios\Lab2Redes\Hamming> ./Receptor
Ingrese el mensaje binario con los bits de paridad generados por el emisor: 01111101001111
Se detectaron y corrigieron errores.
Posición del bit a corregir: 2
Mensaje codificado corregido: 00111101001111
Mensaje corregido: 1110001111
```

**Mensaje 3 - 1001011001101**

- **Mensaje codificado - 0111001001100111101**
- **Mensaje modificado - 0111001101100111001**

```
PS C:\Documentos\Semestre8\Redes\Laboratorios\Lab2Redes\Hamming> python .\Emisor.py
Ingrese el mensaje a codificar: 1001011001101
Mensaje codificado con Hamming: 0111001001100111101
PS C:\Documentos\Semestre8\Redes\Laboratorios\Lab2Redes\Hamming> ./Receptor
Ingrese el mensaje binario con los bits de paridad generados por el emisor: 0111001101100111001
Se detectaron y corrigieron errores.
Posición del bit a corregir: 8
Mensaje codificado corregido: 0111001001100111001
Mensaje corregido: 1001011001101
```

## Algoritmo de detección de errores: Fletcher checksum

**Mensaje 1 - 1100**

- **Mensaje codificado**
  **00001100000011100000010**
- **Mensaje modificado**
  **00001100000011100000100**

```
PS C:\Documentos\Semestre8\Redes\Laboratorios\Lab2Redes\Fletcher_Checksum> python .\Emisor.py
--- Emisor ---
Ingrese el mensaje binario: 1100
Ingrese el tamaño del bloque (8, 16, o 32): 8
Mensaje con checksum: 00001100000011100000010
Checksum en binario: 0000011100000010
PS C:\Documentos\Semestre8\Redes\Laboratorios\Lab2Redes\Fletcher_Checksum> ./Receptor

--- Receptor ---
Ingrese el mensaje binario con el checksum: 00001100000011100000100
Ingrese el tamaño del bloque (8, 16, o 32): 8
Se detectaron errores en el mensaje. Se descarta el mensaje ❌
```

**Mensaje 2 - 1110001110**

- **Mensaje codificado**
  **00000011100011100000000000010010000000000000000110**
- **Mensaje modificado**
  **00000011100011100000000000010010000000000000100111**

```
PS C:\Documentos\Semestre8\Redes\Laboratorios\Lab2Redes\Fletcher_Checksum> python .\Emisor.py
--- Emisor ---
Ingrese el mensaje binario: 1110001110
Ingrese el tamaño del bloque (8, 16, o 32): 16
Mensaje con checksum: 00000011100011100000000000010010000000000000000110
Checksum en binario: 0000000000100100000000000000110
PS C:\Documentos\Semestre8\Redes\Laboratorios\Lab2Redes\Fletcher_Checksum> ./Receptor

--- Receptor ---
Ingrese el mensaje binario con el checksum: 00000011100011100000000000010010000000000000100111
Ingrese el tamaño del bloque (8, 16, o 32): 16
Se detectaron errores en el mensaje. Se descarta el mensaje ❌
```

**Mensaje 3 - 1001011001101**

- **Mensaje codificado**
  **00000000000000000001001011001101000000000000000000000000000001011100000000000000000000000000111**
- **Mensaje modificado**
  **00000000000000000001001011001101000000000000000000000000000001011100000000000000000000000001110**

```
PS C:\Documentos\Semestre8\Redes\Laboratorios\Lab2Redes\Fletcher_Checksum> python .\Emisor.py
--- Emisor ---
Ingrese el mensaje binario: 1001011001101
Ingrese el tamaño del bloque (8, 16, o 32): 32
Mensaje con checksum: 00000000000000000001001011001101000000000000000000000000000001011100000000000000000000000000111
Checksum en binario: 00000000000000000000000001011100000000000000000000000000111
PS C:\Documentos\Semestre8\Redes\Laboratorios\Lab2Redes\Fletcher_Checksum> ./Receptor

--- Receptor ---
Ingrese el mensaje binario con el checksum: 00000000000000000001001011001101000000000000000000000000000001011100000000000000000000000001110
Ingrese el tamaño del bloque (8, 16, o 32): 32
Se detectaron errores en el mensaje. Se descarta el mensaje ❌
```

➔ **¿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? En caso afirmativo, demuestrelo con su implementación.**

Tanto en Hamming como en Fletcher Checksum podemos ver que cuando se agregó un error Hamming pudo corregir el mensaje bien dado tanto el mensaje codificado como el original sin errores y Checksum pudo detectar bien el error.

En el caso donde se agregaron más de un error podemos ver que Checksum si lo hizo ya que solo era de detectarlo, pero Hamming detectó solo un error, por lo que al devolver el mensaje original como el codificado no los devolvió correctamente.

Por ello en Hamming si es posible manipular los bits de tal manera que no sea capaz de detectar todos los errores, ya que solo detectara uno y ese corregirá pero los demás no será capaz. Sin embargo con Checksum ya que es un algoritmo de detección de errores entonces será capaz de detectarlos sin importar la cantidad aunque no los corrija.

➔ **En base a las pruebas que realizó, ¿qué ventajas y desventajas posee cada algoritmo con respecto a los otros dos? Tome en cuenta complejidad, velocidad, redundancia (overhead), etc. Ejemplo: "En la implementación del bit de paridad par, me di cuenta que comparado con otros métodos, la redundancia es la mínima (1 bit extra). Otra ventaja es la facilidad de implementación y la velocidad de ejecución, ya que se puede obtener la paridad aplicando un XOR entre todos los bits. Durante las pruebas, en algunos casos el algoritmo no era capaz de detectar el error, esto es una desventaja, por ejemplo [...]."**

Con Hamming algunas de sus ventajas es que es capaz de detectar y corregir el error en el mensaje siendo bastante eficiente, pero su desventaja sería que tiene la limitación de corregir un solo error y falla al momento de presentarse varios de ellos. Al momento de programarlo es considerado uno de los más sencillos de implementar dado que la dificultad solo radica en el cálculo de los bits adicionales que se añaden para la detección y corrección del error.

Ahora con Fletcher Checksum pudimos determinar que sí es capaz de detectar tanto uno como múltiples errores siendo más eficiente. Además, al añadir los bits al mensaje para detectar errores se considera más sencillo ya que solo se debe concatenar la codificación generada al mensaje original. Dentro de sus desventajas encontramos que solo puede detectar los errores y no puede corregirlos. Por otra parte, dado que debe manejar distinta cantidad de bloques (8, 16 o 32), se complicó un poco la implementación para cada codificación.