Universidad Del Valle de Guatemala
Departamento de Computación
Redes

Daniel Esteban Morales Urizar - 21785
Brandon Rolando Sicay Cumes - 2175

Laboratorio 2: Parte 1

Esquemas de detección y corrección de errores

Descripción de la práctica

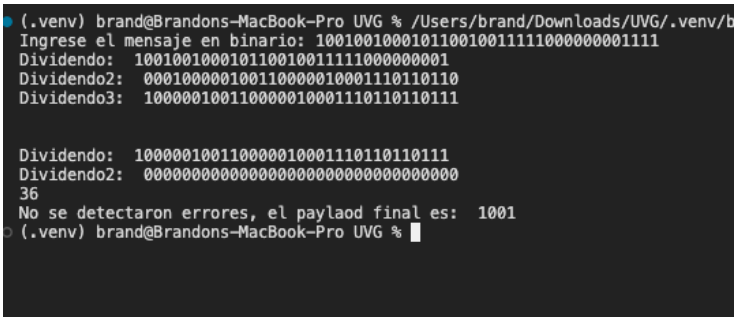
Se implementaron dos algoritmos sobre la detección de errores en mensajes: CRC-32 y Hamming. Para ambos algoritmos, se desarrollaron un emisor y un receptor. El emisor se encargaba de generar una trama, una secuencia de bits (1s y 0s) de cualquier longitud, añadiendo ciertos bits de referencia para identificar errores en la trama. Los receptores ingresaban la trama generada y realizaban operaciones para detectar y, en el caso del código de Hamming, corregir errores. Los emisores fueron programados en Java y los receptores en Python. Además, se modificaron manualmente ciertos bits de las tramas generadas para inducir errores. Se probaron tres tramas sin modificar, tres con un bit modificado, tres con dos bits modificados y una trama especialmente modificada para que el algoritmo no pudiera detectarlo.

El CRC-32 (Cyclic Redundancy Check de 32 bits) es un algoritmo de detección de errores ampliamente utilizado en comunicaciones y almacenamiento de datos. Este se basa en la configuración de un polinomio generador estandarizado, representado de forma binaria. Para codificar el mensaje, se agregan 32 ceros al final y se realiza una “división” usando la función XOR entre el mensaje modificado y el polinomio de CRC-32. El residuo de esta división se agrega en lugar de los ceros, sustituyendolos según el tamaño del residuo. Para verificar la integridad del mensaje, se repite la operación buscando un resultado de división igual a 0. Si el resultado es 0, el mensaje es considerado íntegro; de lo contrario, se detectan errores.

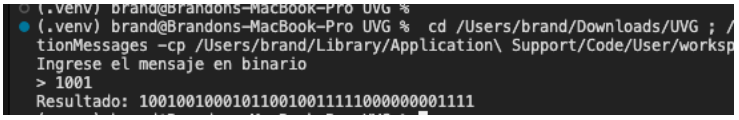
El código de Hamming es un tipo de código de corrección de errores que agrega bits de paridad a los datos originales para detectar y corregir errores en la transmisión y almacenamiento de datos. Los bits de paridad se calculan usando reglas específicas y se colocan en ciertas posiciones dentro de la secuencia de datos, generalmente en posiciones que son potencias de dos. Durante la transmisión o almacenamiento, los bits de paridad permiten detectar errores y, en algunos casos, corregirlos. Dependiendo del tipo de código de Hamming utilizado, el algoritmo puede corregir errores de un solo bit o solo detectarlos. Si se detecta un error, es posible determinar y corregir el bit incorrecto usando los bits de paridad.

Resultados

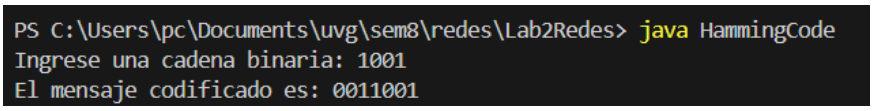
- Tramas Utilizadas
    - Correctas
        - 1001
            - CRC-32
                - Emisor



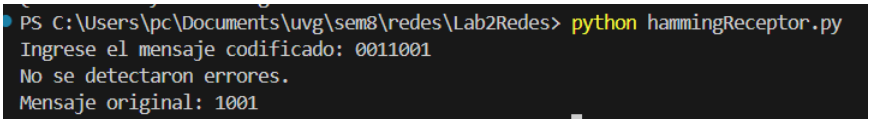
                - Receptor



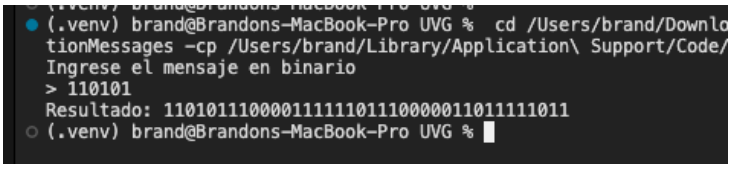
            - Hamming
                - Emisor



                - Receptor

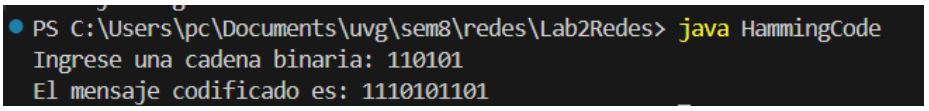


- 110101
  - CRC-32
    - Emisor

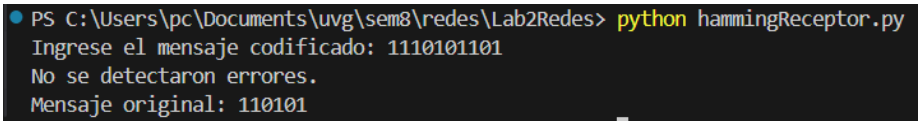


    - Receptor

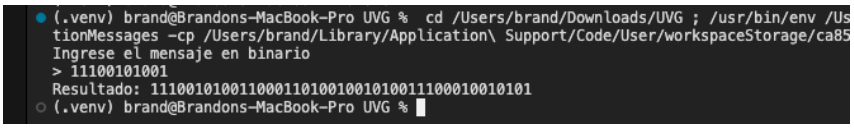
  - Hamming
    - Emisor



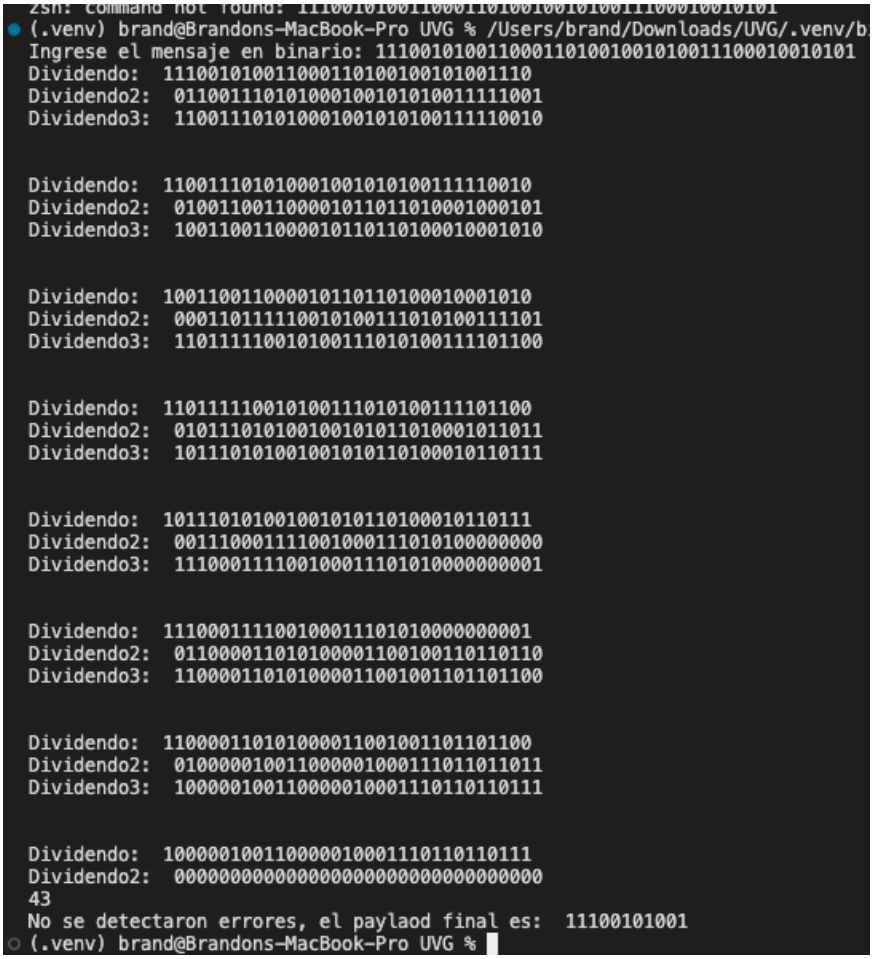
    - Receptor



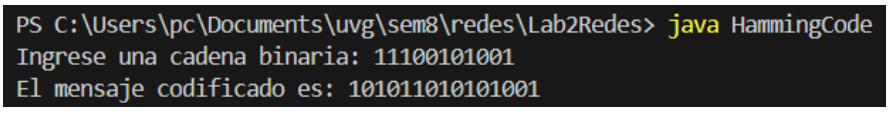
- 11100101001
  - CRC-32
    - Emisor



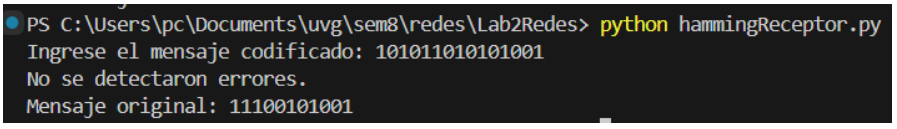
    - Receptor



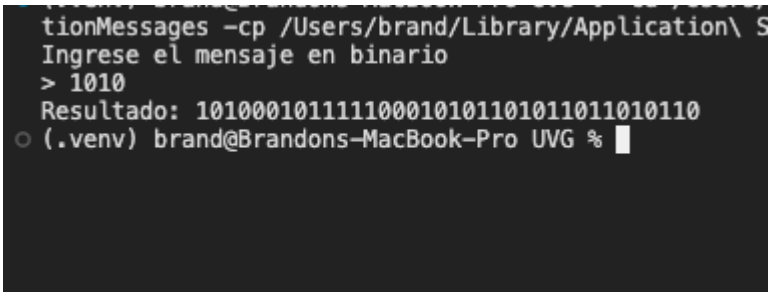
  - Hamming
    - Emisor



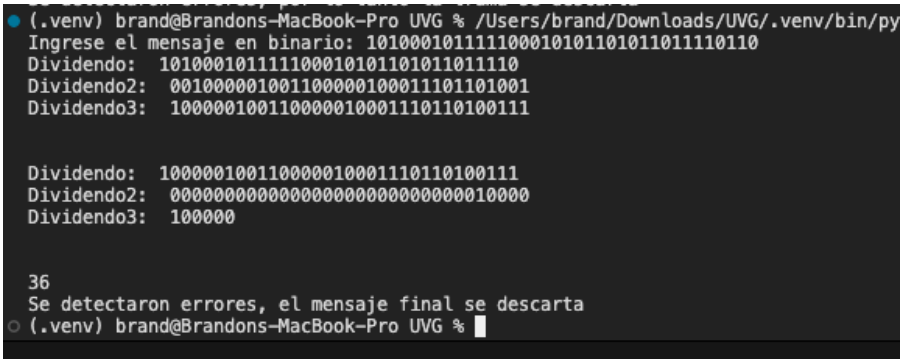
    - Receptor



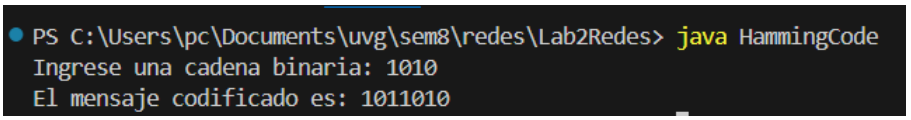
- 1 bit modificado
  - Bit ingresado: 1010
    - CRC-32



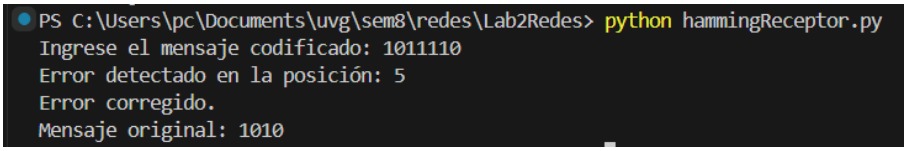
      - Output modificado: 10100010111110001010110101101110110
      - Receptor



    - Hamming



      - Output Modificado: 1011110
      - Receptor



- Bit ingresado: 10010
  - CRC-32

```
tionMessages -cp /Users/brand/Library/Application\ Support/Code/User/wo
Ingrese el mensaje en binario
> 10010
Resultado: 10010010001011001001111100000000011110
(.venv) brand@Brandons-MacBook-Pro UVG %
```

    - Output modificado: 10010010001011001011111100000000011110
    - Receptor

```
(.venv) brand@Brandons-MacBook-Pro UVG % /Users/brand/Downloads/UVG/.ve
Ingrese el mensaje en binario: 10010010001011001011111100000000011110
Dividendo:  10010010001011001011111100000000001
Dividendo2:  00010000010011000011000111011011O
Dividendo3:  10000010011000011000111011011O111


Dividendo:  10000010011000011000111011011O111
Dividendo2:  00000000000000010000000000000000
Dividendo3:  100000000000000000


37
Se detectaron errores, el mensaje final se descarta
(.venv) brand@Brandons-MacBook-Pro UVG %
```

  - Hamming

```
PS C:\Users\pc\Documents\uvg\sem8\redes\Lab2Redes> java HammingCode
Ingrese una cadena binaria: 10010
El mensaje codificado es: 001100100
```

    - Output modificado: 001100000
    - Receptor

```
PS C:\Users\pc\Documents\uvg\sem8\redes\Lab2Redes> python hammingReceptor.py
Ingrese el mensaje codificado: 001100000
Error detectado en la posición: 7
Error corregido.
Mensaje original: 10010
```

- Bit ingresado: 11000
  - CRC-32

```
Ingrese el mensaje en binario
> 11000
Resultado: 11000011010100001100100111011011001000
(.venv) brand@Brandons-MacBook-Pro UVG %
```

    - Output modificado: 11000011010100001100100111011011011000
    - Receptor

```
Ingrese el mensaje en binario: 11000011010100001100100111011011011000
Dividendo:  11000011010100001100100111011011
Dividendo2:  01000001001100000100011101101010
Dividendo3:  10000010011000001000111011010101


Dividendo:  10000010011000001000111011010101
Dividendo2:  00000000000000000000000000000010
Dividendo3:  10000


37
Se detectaron errores, el mensaje final se descarta
(.venv) brand@Brandons-MacBook-Pro UVG %
```

  - Hamming

```
PS C:\Users\pc\Documents\uvg\sem8\redes\Lab2Redes> java HammingCode
Ingrese una cadena binaria: 11000
El mensaje codificado es: 011110000
```

    - Output modificado: 011110001
    - Receptor

```
PS C:\Users\pc\Documents\uvg\sem8\redes\Lab2Redes> python hammingReceptor.py
Ingrese el mensaje codificado: 011110001
Error detectado en la posición: 9
Error corregido.
Mensaje original: 11000
```

- 2 bits modificados
  - Bit ingresado: 1000
    - CRC-32

```
tionMessages -cp /Users/brand/Library/Application\ S
Ingrese el mensaje en binario
> 1000
Resultado: 10000010011000001000111011011011000
```

      - Output modificado: 10000010011100001000111011110111000
      - Receptor

```
Ingrese el mensaje en binario: 10000010011100001000111011111011000
Dividendo:  10000010011100001000111011111011
Dividendo2:  00000000001000000000000001000000
Dividendo3:  10000000000000001000000000


36
Se detectaron errores, el mensaje final se descarta
```

    - Hamming

```
PS C:\Users\pc\Documents\uvg\sem8\redes\Lab2Redes> java HammingCode
Ingrese una cadena binaria: 1000
El mensaje codificado es: 1110000
```

      - Output modificado: 1010010
      - Receptor

```
PS C:\Users\pc\Documents\uvg\sem8\redes\Lab2Redes> python hammingReceptor.py
Ingrese el mensaje codificado: 1010010
Error detectado en la posición: 4
Error corregido.
Mensaje original: 1010
```

Hamming no puede detectar dos o más errores debido a la forma en que calcula y verifica los bits de paridad.

- Bit ingresado: 1010101
  - CRC-32

```
tionMessages -cp /Users/brand/Library/Application\ Sup
Ingrese el mensaje en binario
> 1010101
Resultado: 10101010110111101010010110000000101101100
(.venv) brand@Brandons-MacBook-Pro UVG %
```

    - Output modificado: 10101010110111111010010110010000101101100
    - Receptor

```
(.venv) brand@Brandons-MacBook-Pro UVG % /Users/brand/Downloads/UVG/.ven
Ingrese el mensaje en binario: 10101010110111111010010110010000101101100
Dividendo:  10101010110111111010010110010000101101100
Dividendo2:  00101000101111110010101101001011010
Dividendo3:  10100010111111001010110100101101 0


Dividendo:  10100010111111001010110100101101 0
Dividendo2:  00100000100111000010001111110110 1
Dividendo3:  10000010011100001000111111011011 1


Dividendo:  10000010011100001000111111011011 1
Dividendo2:  00000000000100000000000100000000 0
Dividendo3:  10000000000100000000000


39
Se detectaron errores, el mensaje final se descarta
(.venv) brand@Brandons-MacBook-Pro UVG %
```

  - Hamming

```
PS C:\Users\pc\Documents\uvg\sem8\redes\Lab2Redes> java HammingCode
Ingrese una cadena binaria: 1010101
El mensaje codificado es: 11110100101
```

    - Output modificado: 10110100111
    - Receptor

```
PS C:\Users\pc\Documents\uvg\sem8\redes\Lab2Redes> python hammingReceptor.py
Ingrese el mensaje codificado: 10110100111
Error detectado en la posición: 8
Error corregido.
Mensaje original: 1010111
```

Hamming no puede detectar dos o más errores debido a la forma en que calcula y verifica los bits de paridad.

- Bit ingresado: 10000110001
  - CRC-32

```
tionMessages -cp /Users/brand/Library/Application\ Sup
Ingrese el mensaje en binario
> 1010101
Resultado: 10101010110111101010010110000000101101100
(.venv) brand@Brandons-MacBook-Pro UVG %
```

    - Output modificado: 10101010110111111010010110010000101101100
    - Receptor

```
Ingrese el mensaje en binario: 10101010110111111010010110010000101101100
Dividendo:  10101010110111111010010110010000101101100
Dividendo2:  00101000101111110010101101001011010
Dividendo3:  10100010111111001010110100101101 0


Dividendo:  10100010111111001010110100101101 0
Dividendo2:  00100000100111000010001111110110 1
Dividendo3:  10000010011100001000111111011011 1


Dividendo:  10000010011100001000111111011011 1
Dividendo2:  00000000000100000000000100000000 0
Dividendo3:  10000000000010000000000


39
Se detectaron errores, el mensaje final se descarta
```

  - Hamming

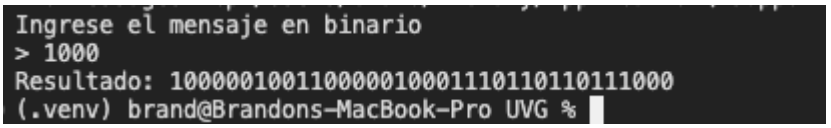
```
PS C:\Users\pc\Documents\uvg\sem8\redes\Lab2Redes> java HammingCode
Ingrese una cadena binaria: 10000110001
El mensaje codificado es: 101100010110001
```

    - Output modificado: 1101100010110001
    - Receptor

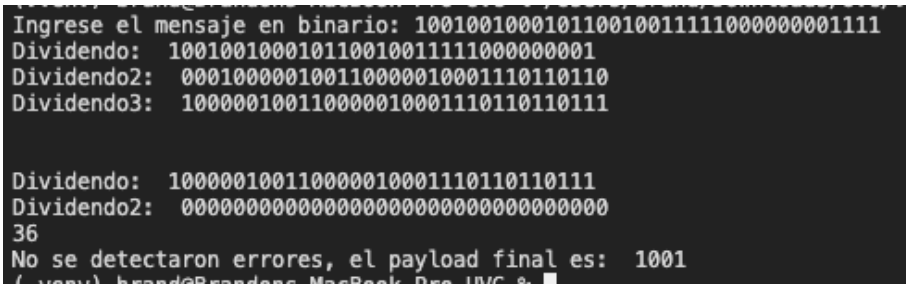
```
PS C:\Users\pc\Documents\uvg\sem8\redes\Lab2Redes> python hammingReceptor.py
Ingrese el mensaje codificado: 110100010110001
Error detectado en la posición: 1
Error corregido.
Mensaje original: 00000110001
```

Hamming no puede detectar dos o más errores debido a la forma en que calcula y verifica los bits de paridad.

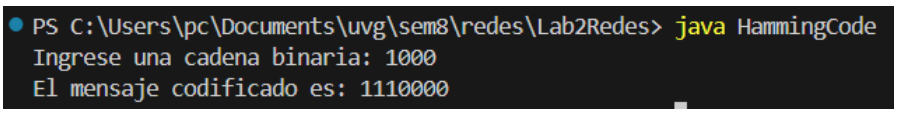
- Manipulación de bits para ser incapaz de detectar error
  - CRC-32
    - Trama utilizada: 1000



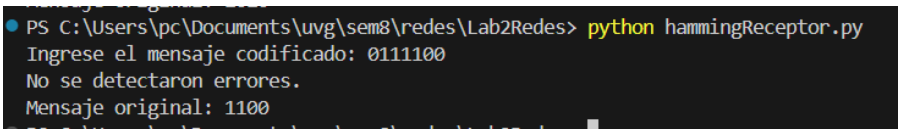
    - Output modificado: 10010010001011001001111100000000 1111
    - Receptor:



  - Hamming
    - Trama utilizada: 1000



    - Output modificado: 0111100
    - Receptor:



Discusión

- ¿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
  - En el algoritmo CRC-32, la manera más sencilla de lograr que una trama pase, sin detectar ningún error es hacer que una trama válida A, se convierta en otra trama válida B, con un error en específico. Específicamente, se tomaron las tramas 1000 para A, y para B sería 1001. La trama codificada de A sería 10000010011000001000111011011011000, mientras que la de B sería 10010010001011001001111100000000 1111, por lo que, al simular "ruido", hace que los bits tengan flips; y es por ello que en base a la trama codificada de B, se modificó la trama de A para que quedara de

esta manera 10010010001011001001111100000000 1111. Es decir, la transformación del primer mensaje debe de tomar la forma de otra trama codificada correctamente.

- En el caso de Hamming, sí es posible manipular los bits de tal manera que el código de Hamming no detecte el error, ya que está diseñado para corregir solo un error. Si se manipulan dos o más bits de manera específica, los bits de paridad pueden dar una suma de verificación que no refleja ninguna posición incorrecta o que apunta a una posición incorrecta, resultando en la incapacidad del algoritmo para detectar o corregir adecuadamente los errores múltiples.

Conclusiones

- Existen diversas formas de codificar y decodificar mensajes binarios, cada uno con sus ventajas y desventajas.
- El algoritmo CRC-32 de detección de errores es más sencillo de implementar que el algoritmo de corrección de errores Hamming.
- Hamming no puede detectar dos o más errores debido a la forma en que calcula y verifica los bits de paridad.

Referencias

Grover, A., & Singh, S. (2015). Comparative Analysis of CRC-32 and SHA-1 Algorithms in WEP. Advanced Engineering Technology and Application, 4(1), 5-10.

Griffiths, G., & Stones, G. C. (1987). The tea-leaf reader algorithm: an efficient implementation of CRC-16 and CRC-32. Communications of the ACM, 30(7), 617-620.