Redes - Sección 20
Melissa Pérez - 21385
Fernanda Esquivel - 21542

Laboratorio #2

Esquemas de detección y corrección de errores

Enlace al repositorio

Puede acceder al repositorio haciendo clic aquí.

Algoritmos utilizados

1. Corrección de errores: códigos de Hamming para cualquier combinación (n,m) válida
2. Detección de errores: CRC-32

Códigos de Hamming

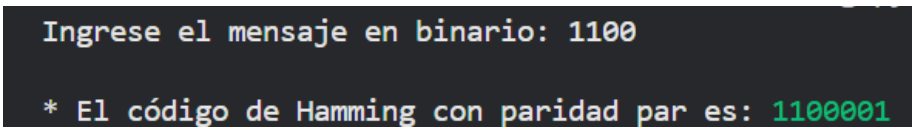
- Enviar un mensaje al emisor, copiar el mensaje generado por este y proporcionarlo tal cual al receptor, el cual debe mostrar el mensajes originales (ya que ningún bit sufrió un cambio). Realizar esto para tres mensajes distintos con distinta longitud.
    - 1010
        - Emisor

```
PS C:\Users\Usuario\Documents\GitHub\RS-Lab-02> & C
io/Documents/GitHub/RS-Lab-02/src/emisor/Hamming.py
Ingrese el mensaje en binario: 1010

* El código de Hamming con paridad par es: 1010010
```

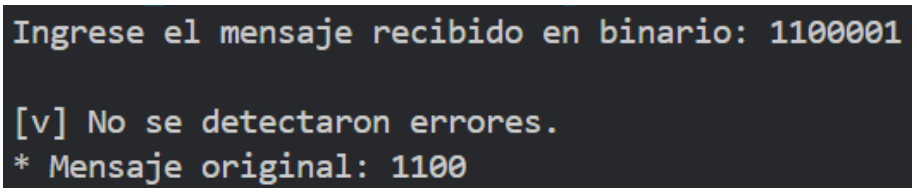
        - Receptor

```
-Lab-02_998f66b9\bin' 'Hamming'
Ingrese el mensaje recibido en binario: 1010010

[v] No se detectaron errores.
* Mensaje original: 1010
```

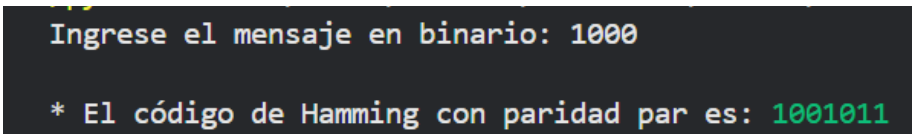
- 1100
  - Emisor



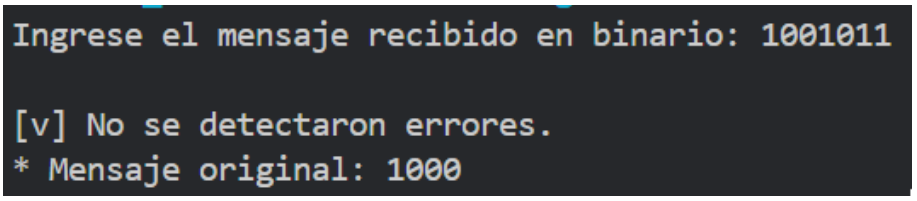
  - Receptor



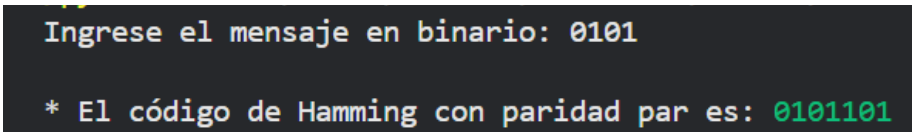
- 1000
  - Emisor



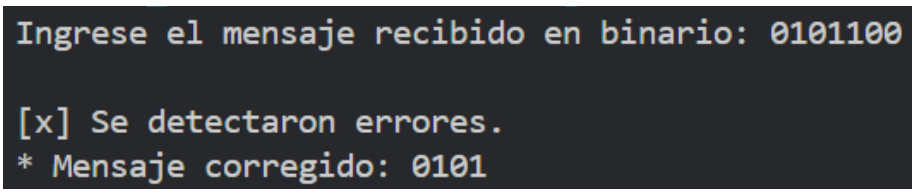
  - Receptor



- 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.
  - 0101; cambiar bit final por 0
    - Emisor



    - Receptor



- 0011; cambiar bit final por 1
  - Emisor

```
  Ingrese el mensaje en binario: 0011

  * El código de Hamming con paridad par es: 0011110
```

  - Receptor

```
Ingrese el mensaje recibido en binario: 0011111

[x] Se detectaron errores.
* Mensaje corregido: 0011
```

- 0001; cambiar bit final por 0
  - Emisor

```
Ingrese el mensaje en binario: 0001

* El código de Hamming con paridad par es: 0000111
```

  - Receptor

```
Ingrese el mensaje recibido en binario: 0000110

[x] Se detectaron errores.
* Mensaje corregido: 0001
```

- 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.
  - NOTA: El algoritmo de Hamming solo puede corregir un error.

  - 1111; cambiar bits finales por 00
    - Emisor

```
PS C:\Users\Usuario\Documents\GitHub\RS-Lab-02\src>
/python.exe c:/Users/Usuario/Documents/GitHub/RS-Lab
Ingrese el mensaje en binario: 1111

* El código de Hamming con paridad par es: 1111111
```

- Receptor

```
-Lab-02_998f66b9\bin' 'Hamming'
Ingrese el mensaje recibido en binario: 1111100

[x] Se detectaron errores.
* Mensaje corregido: 1110
```

- 0000; cambiar bits finales por 11
  - Emisor

```
PS C:\Users\Usuario\Documents\GitHub\RS-Lab-02\src>
/python.exe c:/Users/Usuario/Documents/GitHub/RS-La
Ingrese el mensaje en binario: 0000

* El código de Hamming con paridad par es: 0000000
```

  - Receptor

```
-Lab-02_998f66b9\bin' 'Hamming'
Ingrese el mensaje recibido en binario: 0000011

[x] Se detectaron errores.
* Mensaje corregido: 0001
```

- 1110; cambiar bits finales por 11
  - Emisor

```
PS C:\Users\Usuario\Documents\GitHub\RS-Lab-02\src>
/python.exe c:/Users/Usuario/Documents/GitHub/RS-Lab
Ingrese el mensaje en binario: 1110

* El código de Hamming con paridad par es: 1111000
```

  - Receptor

```
-Lab-02_998f66b9\bin' 'Hamming'
Ingrese el mensaje recibido en binario: 1111011

[x] Se detectaron errores.
* Mensaje corregido: 1111
```

- ¿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.
    - Sí es posible, pues el código de Hamming está diseñado para detectar y corregir errores simples (un solo bit erróneo) y para detectar, pero no corregir, errores dobles (dos bits erróneos).
    - Esto puede verse reflejado en las respuestas a las dos preguntas anteriores, donde en el caso de cambiar un solo bit el algoritmo detecta y corrige el error sin problemas, sin embargo, al cambiar dos bits el algoritmo ya no es capaz de corregir el error correctamente (solamente lo detecta).

- 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.
    - Comparación con el código de paridad

      Ventajas:
        - Los códigos de Hamming no solo detectan errores en los mensajes, sino que también los corrigen (siempre y cuando el error sea de un solo bit). Mientras que, el código de paridad generalmente solo puede detectar un número impar de errores y no puede corregirlos.

      Desventajas:
        - Los códigos de Hamming requieren más bits de paridad que el código de paridad, lo que significa que tiene un mayor coste computacional a la hora de corregir los errores.
        - Los códigos de Hamming son más complejos de implementar que el código de paridad.

    - Comparación con los códigos convolucionales

      Ventajas:
        - A pesar de que los códigos de Hamming ya tienen cierto coste computacional, en general, estos requieren menos recursos computacionales para la codificación y decodificación en comparación con los códigos convolucionales.
        - Asimismo, debido a su menor complejidad, los códigos de Hamming pueden ser más rápidos en términos de procesamiento que los códigos convolucionales.

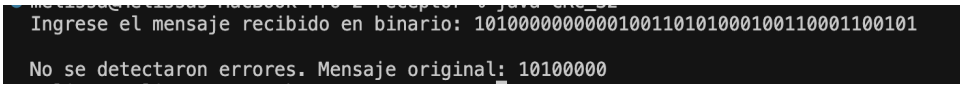
Desventajas:

- Los códigos de Hamming están limitados a la corrección sólo un bit erróneo y la detección de hasta dos errores. Los códigos convolucionales, especialmente cuando se utilizan con el algoritmo de Viterbi, pueden manejar la corrección de varios errores.

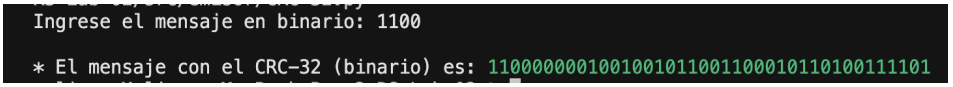
Algoritmo CRC-32

- 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.
  - 1010
    - Emisor

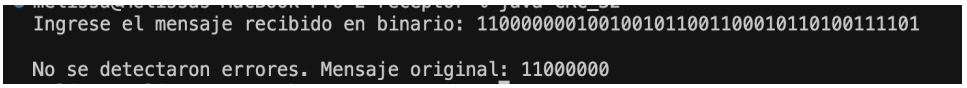
    - Receptor



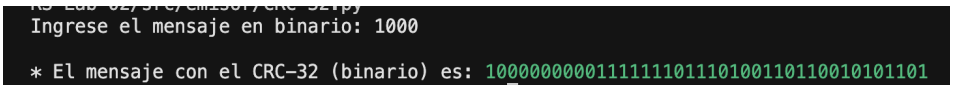
  - 1100
    - Emisor



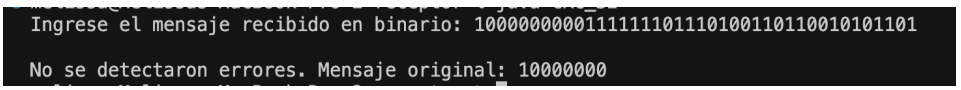
    - Receptor



  - 1000
    - Emisor



    - Receptor



- 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.

- 0101; cambiar bit final por 0
  - Emisor

```
Ingrese el mensaje en binario: 0101

* El mensaje con el CRC-32 (binario) es: 01010000101110010110100110111110011111001
```

  - Receptor

```
Ingrese el mensaje recibido en binario: 01010000101110010110100110111110011111000

Se detectaron errores. Se descarta el mensaje.
```

- 0011; cambiar bit final por 1
  - Emisor

```
Ingrese el mensaje en binario: 0011

* El mensaje con el CRC-32 (binario) es: 00110000111101001101101111011111100100001
```

  - Receptor

```
Ingrese el mensaje recibido en binario: 00110000111101001101101111011111100100011

Se detectaron errores. Se descarta el mensaje.
```

- 0001; cambiar bit final por 0
  - Emisor

```
Ingrese el mensaje en binario: 0001

* El mensaje con el CRC-32 (binario) es: 00010000110011111011010111111111111101001
```

  - Receptor

```
Ingrese el mensaje recibido en binario: 00010000110011111011010111111111111101000

Se detectaron errores. Se descarta el mensaje.
```

- 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.
  - 1111; cambiar bits finales por 00
    - Emisor

```
Ingrese el mensaje en binario: 1111

* El mensaje con el CRC-32 (binario) es: 11110000011011110111111000111011001000 1
```

        - Receptor

```
Ingrese el mensaje recibido en binario: 11110000011011111011111100011101100100000

Se detectaron errores. Se descarta el mensaje.
```

    - 0000; cambiar bits finales por 11
        - Emisor

```
Ingrese el mensaje en binario: 0000

* El mensaje con el CRC-32 (binario) es: 00000000110100100000001011101111100011101
```

        - Receptor

```
Ingrese el mensaje recibido en binario: 00000000110100100000001011101111100011111

Se detectaron errores. Se descarta el mensaje.
```

    - 1110; cambiar bits finales por 11
        - Emisor

```
Ingrese el mensaje en binario: 1110

* El mensaje con el CRC-32 (binario) es: 11100000011100100000100000001101111101010
```

        - Receptor

```
Ingrese el mensaje recibido en binario: 11100000011100100000100000001101111101111

Se detectaron errores. 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.
    - Teóricamente, sí es posible manipular los bits para que CRC-32 no detecte un error, eso sucede porque está diseñado para detectar errores con cambios en un solo bit. Por ejemplo, al alterar dos bits en posiciones específicas de forma que una cancele la modificación de otra en beneficio al cálculo del algoritmo, el error no será detectado.
    - En nuestra implementación, no es posible realizar esa manipulación porque se mantiene el mismo checksum para conservar el valor de la cadena.

- 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.
    - Ventajas
        - Tiene una baja redundancia comparado con los métodos de corrección de errores.

        - Alta probabilidad de detección de errores comunes como de un solo bit.
    - Desventajas
        - No puede corregir errores, únicamente los detecta.
        - Posibilidad de falsos negativos con patrones específicos como se ha respondido en la pregunta anterior.