Universidad Del Valle de Guatemala — Departamento de Computación
Redes — Sección 20
Elías Alberto Alvarado Raxón — 21808
Adrian Fulladolsa Palma — 21592

Laboratorio #2 - Esquemas de detección y corrección - Parte 1

Implementación de algoritmos

Los algoritmos seleccionados para esta práctica fueron el Código de Hamming para cualquier combinación (n,m) válida como el algoritmo de corrección de errores y el algoritmo de Fletcher Checksum en su variante de Fletcher-16 como el algoritmo de detección de errores.

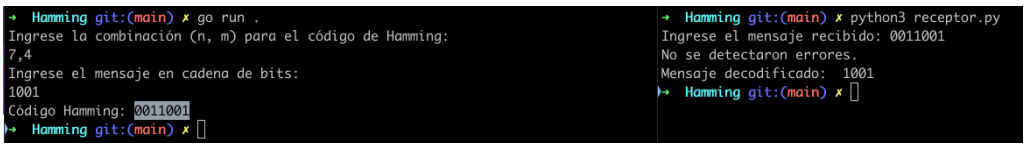
Los lenguajes de programación seleccionados fueron Python y Go o GoLang. El emisor del algoritmo de Código de Hamming fue desarrollado en Go y su receptor en Python, de manera opuesta, el emisor del algoritmo de Fletcher Checksum fue desarrollado en Python y su receptor en Go.

Mensajes seleccionados

- 1001
- 0100100
- 100110

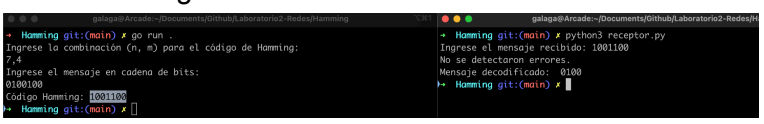
Escenarios de prueba

1. Envió de mensajes sin cambios
    - 1001
        - Hamming



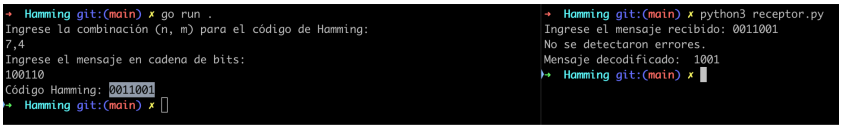
        - Fletcher-16

    - 0100100
        - Hamming



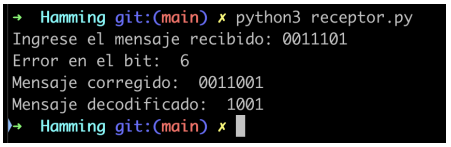
- Fletcher-16

- 100110
  - Hamming

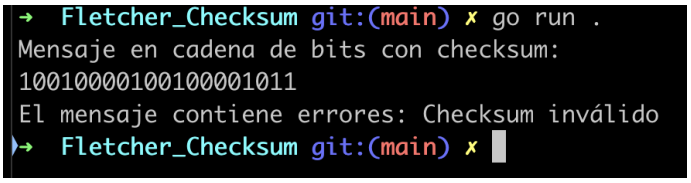


  - Fletcher-16

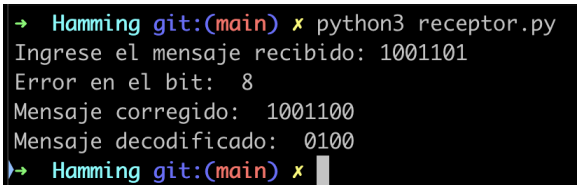
2. Envió de mensajes con cambios a un bit
   - 1001
     - Hamming
       - Mensaje utilizado: 00011101
       - Bit cambiado 6



     - Fletcher-16
       - Mensaje utilizado: 10010000100100001011
       - Bit cambiado 19



   - 0100100
     - Hamming
       - Mensaje utilizado: 01001101
       - Bit modificado: 8



     - Fletcher-16
       - Mensaje utilizado: 0100100001001000010010 1
       - Bit modificado: 23

```
➜  Fletcher_Checksum git:(main) ✗ go run .
Mensaje en cadena de bits con checksum:
01001000010010000100101
El mensaje contiene errores: Checksum inválido
➜  Fletcher_Checksum git:(main) ✗
```

- 100110
    - Hamming
        - Mensaje utilizado: 01011001
        - Bit modificado: 2

```
➜  Hamming git:(main) ✗ python3 receptor.py
Ingrese el mensaje recibido: 1011001
Error en el bit:  2
Mensaje corregido:  0011001
Mensaje decodificado:  1001
➜  Hamming git:(main) ✗
```

    - Fletcher-16
        - Mensaje utilizado: 100110001000100100110
        - Bit modificado: 12

```
➜  Fletcher_Checksum git:(main) ✗ go run .
Mensaje en cadena de bits con checksum:
100110001000100100110
El mensaje contiene errores: Checksum inválido
➜  Fletcher_Checksum git:(main) ✗
```

3. Envió de mensajes con cambios a dos o más bits
    - 1001
        - Hamming
            - Mensaje utilizado: 01011101
            - Bits modificados: 2 y 8

```
➜  Hamming git:(main) ✗ python3 receptor.py
Ingrese el mensaje recibido: 1011101
Error en el bit:  5
Mensaje corregido:  1010101
Mensaje decodificado:  1101
➜  Hamming git:(main) ✗
```

        - Fletcher-16
            - Mensaje utilizado: 10011000100100001011
            - Bits modificados: 5 y 19

```
➜  Fletcher_Checksum git:(main) ✗ go run .
Mensaje en cadena de bits con checksum:
10011000100100001011
El mensaje contiene errores: Checksum inválido
➜  Fletcher_Checksum git:(main) ✗
```

- 0100100
  - Hamming
    - Mensaje utilizado: 01101101
    - Bits modificados: 3 y 8

```
→  Hamming git:(main) ✗ python3 receptor.py
Ingrese el mensaje recibido: 01101101
Error en el bit:  11
Mensaje es descartado
Mensaje decodificado:  1110
→  Hamming git:(main) ✗
```

  - Fletcher-16
    - Mensaje utilizado: 11001000010010000100101
    - Bits modificados: 1 y 23

```
→  Fletcher_Checksum git:(main) ✗ go run .
Mensaje en cadena de bits con checksum:
01001000010010000100100
El mensaje fue aceptado: 0100100
 %
→  Fletcher_Checksum git:(main) ✗
```

- 100110
  - Hamming
    - Mensaje utilizado: 01010001
    - Bits modificados: 1 y 5

```
→  Hamming git:(main) ✗ python3 receptor.py
Ingrese el mensaje recibido: 10010001
Error en el bit:  14
Mensaje es descartado
Mensaje decodificado:  0000
→  Hamming git:(main) ✗
```

  - Fletcher-16
    - Mensaje utilizado: 01010001
    - Bits modificados: 1 y 5

```
→  Fletcher_Checksum git:(main) ✗ go run .
Mensaje en cadena de bits con checksum:
10011000100010101001l0
El mensaje contiene errores: Checksum inválido
→  Fletcher_Checksum git:(main) ✗
```

4. ¿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 el algoritmo de Código de Hamming sí es posible manipular los bits de tal manera que el mensaje sea aceptado, esto es posible si se alteran tres o más bits. Esto se debe a que si se manipula una cantidad impar de bits la paridad puede ser manipulada para que sea válida. El ejemplo más sencillo para demostrar esto es si un mensaje obtenido paso por una manipulación

donde todos los bits fueron convertidos a 0:

```
➜  Hamming git:(main) ✗ python3 receptor.py
Ingrese el mensaje recibido: 0000000
No se detectaron errores.
Mensaje decodificado:  0000
➜  Hamming git:(main) ✗
```

- En el algoritmo de Fletcher Checksum es posible manipular bits del mensaje y del checksum, en otras palabras al menos un bit de los últimos 16 carácteres del mensaje codificado y al menos un bit del mensaje de tal manera que el checksum generado corresponda con el mensaje alterado. Dentro de las pruebas realizadas se obtuvo este resultado al modificar el primer y el último bit del mensaje con el checksum:

```
➜  Fletcher_Checksum git:(main) ✗ go run .
Mensaje en cadena de bits con checksum:
010010000100100001001000
El mensaje fue aceptado: 0100100
 %
➜  Fletcher_Checksum git:(main) ✗
```

5. En base a las pruebas que realizó, ¿qué ventajas y desventajas posee cada algoritmo con respecto a los otros dos?

	Hamming (n,m)	**Fletcher-16**
Ventajas	• Hamming es capaz de corregir errores cuando un bit está incorrecto. • Eficiencia en cuanto a la información que este algoritmo añade al mensaje original.	• Detecta errores carreados • Es eficiente, incluso en ambientes con recursos limitados.
Desventajas	• Limitado a poder corregir un solo error. • Si el escenario estudiado es muy específico, puede darse el caso que el error no se detecte, y por ende, no se pueda corregir. • Sobrecarga de redundancia, en ocasiones añade cierta redundancia en sus bits de información.	• Solamente detecta errores, pero es incapaz de corregirlo. • Si se presenta un tipo de error muy específico, será incapaz de detectarlo debido al swapping. • Para nuestro caso, el checksum seleccionado es relativamente bajo.

Github: Enlace