

**UNIVERSIDAD DEL VALLE DE GUATEMALA**  
Redes  
Sección 21  
Jorge Yass



## **Laboratorio 2**

Parte 1

Oscar Esteban Donis Martínez - 21610  
Abner Iván García Alegría - 21285

## Algoritmos Utilizados:

- Corrección de errores
  - Códigos de Hamming para cualquier combinación (n,m) válida

### Emisor en C

```
Receptor.py  C Emisor.c  x
Redes > Laboratorio2-Redes > Hamming > C Emisor.c > arrayToBits(int *, char *, int)
5 // Función para convertir una cadena de bits a un array de enteros
6 void bitsToArray(const char* bits, int* array, int len) {
7     for (int i = 0; i < len; i++) {
8         array[i] = bits[i] - '0';
9     }
10 }
11
12 // Función para convertir un array de enteros a una cadena de bits
13 void arrayToBits(int* array, char* bits, int len) {
14     for (int i = 0; i < len; i++) {
15         bits[i] = array[i] + '0';
16     }
17     bits[len] = '\0';
18 }
19
20 // Función para calcular los bits de paridad y formar el código Hamming de 7 bits
21 void hamming74Encoder(int* data_bits, int* hamming_code) {
22     int d1 = data_bits[0];
23     int d2 = data_bits[1];
24     int d3 = data_bits[2];
25     int d4 = data_bits[3];
26
27     // Calcular bits de paridad
28     int p1 = d1 ^ d2 ^ d3;
29     int p2 = d1 ^ d2 ^ d4;
30     int p3 = d1 ^ d3 ^ d4;
31
32     // Construir el código Hamming de 7 bits
```

```
Receptor.py  C Emisor.c  x
Redes > Laboratorio2-Redes > Hamming > C Emisor.c > arrayToBits(int *, char *, int)
42 void processBits(char* input_bits) {
43     int segment[4] = {0, 0, 0, 0};
44
45     for (int j = 0; j < (end - start); j++) {
46         if (start + j >= 0) {
47             segment[4 - (end - start) + j] = data[start + j];
48         }
49     }
50
51     int hamming[7];
52     hamming74Encoder(segment, hamming);
53
54     // Imprimir los bits de paridad calculados
55     printf("Bits de paridad: %d %d %d\n", hamming[3], hamming[5], hamming[6]);
56
57     char block_output[8];
58     arrayToBits(hamming, block_output, 7);
59     strcat(output_bits, block_output);
60 }
61
62     printf("Código Hamming: %s\n", output_bits);
63 }
64
65 int main() {
66     char input_bits[100];
67     printf("Ingrese los bits de datos: ");
68     scanf("%s", input_bits);
69 }
```

## Receptor en Python

```
1 def hamming_7_4_decoder(hamming_code):
2     if len(hamming_code) != 7:
3         raise ValueError("Input must be a list of 7 bits.")
4
5     # Verificar que los bits sean 0 o 1
6     if any(bit not in (0, 1) for bit in hamming_code):
7         raise ValueError("Solo pueden ser bits 0 o 1.")
8
9     # Asignar bits de datos y paridad a sus posiciones
10    d1, d2, d3, p1, d4, p2, p3 = hamming_code
11
12    # Calcular bits de paridad
13    p1_calc = d1 ^ d2 ^ d3
14    p2_calc = d1 ^ d2 ^ d4
15    p3_calc = d1 ^ d3 ^ d4
16
17    # Determinar la posición del error
18    error_position = (p3 != p3_calc) * 1 + (p2 != p2_calc) * 2 + (p1 != p1_calc) * 4
19
20    # Corregir el error si es necesario
21    if error_position != 0:
22        inverted_index = 7 - error_position
23        hamming_code[inverted_index] ^= 1
24
25    # Extraer bits de datos corregidos
26    data_bits = [hamming_code[0], hamming_code[1], hamming_code[2], hamming_code[4]]
```

```
Receptor.py x C Emisor.c
Redes > Laboratorio2-Redes > Hamming > Receptor.py > hamming_7_4_decoder

35 def process_input_bits(input_bits):
36     len_bits = len(input_bits)
37     if len_bits not in [7, 14]:
38         raise ValueError("Debe ingresar exactamente 7 o 14 bits.")
39
40     input_bits = list(map(int, input_bits))
41     blocks = [input_bits[i:i + 7] for i in range(0, len_bits, 7)]
42
43     corrected_bits = []
44     for i, block in enumerate(blocks):
45         data_bits, parities, error_position = hamming_7_4_decoder(block)
46         block_status = "no hubo error" if error_position == 0 else f"error corregido en la posición {error_position}"
47         corrected_block = block[:]
48         corrected_block[error_position] = data_bits[error_position]
49         corrected_bits.extend(corrected_block)
50
51         print(f"Blöque {i + 1} ({block_status}): {corrected_block}")
52
53     print(f"\nMensaje corregido: {corrected_bits}\n")
54
55     # Solicitan al usuario que ingrese los bits de datos
56     print("*****")
57     print("Bienvenido al algoritmo de Hamming (Receptor)")
58     print("*****")
59     input_bits = input("Ingrese 7 o 14 bits de datos: ")
60
61     process_input_bits(input_bits)
```

- Detección de errores
  - CRC-32

### Emisor en Python

```

def XOR(a, b):
    if a == b:
        return '0'
    else:
        return '1'

def verifyXOROperation(polinomioBits, oldMessage):
    newMessage = []

    for id, bit in enumerate(oldMessage):
        if id < len(polinomioBits):
            newBit = XOR(bit, polinomioBits[id])
            newMessage.append(newBit)
        else:
            newMessage.extend(oldMessage[id:])
            oldMessage = newMessage.copy()
            break

    if (oldMessage != newMessage):
        oldMessage = newMessage.copy()

    return oldMessage

```

```

def addChecksum(message, polinomioBits, grado):
    oldMessage = list(message)
    endFlag = False
    start = False

    while not endFlag:
        if start == False:
            start = True
        else:
            startIndex = oldMessage.index('1')
            tempOldmessage = oldMessage[startIndex:]
            if len(tempOldmessage) < len(polinomioBits):
                tempOldmessage = oldMessage[-len(polinomioBits):]

            oldMessage = tempOldmessage.copy()

        if len(oldMessage) == len(polinomioBits):
            endFlag = True

    oldMessage = verifyXOROperation(polinomioBits, oldMessage)

    return ''.join(oldMessage[-grado:])

```

```

def main():
    polinomioBits = '10000010011000001000111011011011'
    gradoPolinomio = 32

    mensaje = input("Ingrese el mensaje: ")

    for bit in mensaje:
        if bit != '1' and bit != '0':
            print('El mensaje debe ser binario (1s y 0s solamente)')
            return

    mensajeExtendido = mensaje + '0' * gradoPolinomio

    messageWithChecksum = mensaje + addChecksum(mensajeExtendido, polinomioBits, gradoPolinomio)

    print("Mensaje Extendido: ", messageWithChecksum)

if __name__ == "__main__":
    main()

```

## Receptor en C#

```

class Program
{
    static char XOR(char a, char b)
    {
        if (a == b)
        {
            return '0';
        }
        else
        {
            return '1';
        }
    }
}

```

```
1 reference
static string VerifyXOROperation(string polinomioBits, string OldMessage)
{
    string NewMessage = "";

    for (int i = 0; i < OldMessage.Length; i++)
    {
        if (i < polinomioBits.Length)
        {
            char newBit = XOR(OldMessage[i], polinomioBits[i]);
            NewMessage += newBit;
        }
        else
        {
            NewMessage += OldMessage[i..];
            OldMessage = NewMessage;
            break;
        }
    }

    if (OldMessage != NewMessage)
    {
        OldMessage = NewMessage;
    }

    return OldMessage;
}
```

```
1 reference
static string VerifyMessage(string polinomioBits, string OldMessage, int gradoPolinomio)
{
    bool endFlag = false;
    bool start = false;

    while (!endFlag)
    {
        if (start == false)
        {
            start = true;
        }
        else
        {
            int startIndex = OldMessage.IndexOf('1');
            if (startIndex == -1)
            {
                break;
            }
            string tempOldMessage = OldMessage[startIndex..];

            if (tempOldMessage.Length < polinomioBits.Length)
            {
                tempOldMessage = OldMessage[^polinomioBits.Length..];
            }

            OldMessage = tempOldMessage;
        }

        if (OldMessage.Length == polinomioBits.Length)
        {
            endFlag = true;
        }

        OldMessage = VerifyXOROperation(polinomioBits, OldMessage);
    }

    return OldMessage[^gradoPolinomio..];
}
```

```
0 references
static void Main()
{
    string polinomioBits = "100000100110000010001110110110111";
    // string polinomioBits = "1001";

    int gradoPolinomio = 32;

    Console.Write("Ingrese el mensaje: ");

    string message = Console.ReadLine() ?? "11010001";

    string verification = VerifyMessage(polinomioBits, message, gradoPolinomio);

    if (verification.Contains('1'))
    {
        Console.WriteLine("Error: Se descarta el mensaje por contener errores.");
    }
    else
    {
        Console.WriteLine($"El mensaje recibido: {message[..^gradoPolinomio]} no contiene errores.");
    }
}
```

### EMISOR:

1. Solicitar un mensaje en binario.
2. Ejecutar el algoritmo seleccionado y generar la información necesaria para comprobar la integridad del mensaje.
3. Devolver el mensaje en binario concatenado dicha información.

### RECEPTOR:

1. Solicitar un mensaje en binario concatenado con la información generada por el emisor.
2. Ejecutar el algoritmo seleccionado y comprobar la integridad del mensaje.
3. Devolver la siguiente información según corresponda:
  - a. No se detectaron errores: mostrar el mensaje original (sin la información generada por el emisor)
  - b. Se detectaron errores: indicar que el mensaje se descarta por detectar errores.
  - c. Se detectaron y corrigieron errores: indicar que se corrigieron errores, indicar posición de los bits que se corrigieron y mostrar el mensaje corregido.

## Escenarios de pruebas

### Para los dos algoritmos realizar:

- Enviar un mensaje al emisor, copiar el mensaje generado por este y proporcionar 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 Hamming(7,4):

Mensaje 1: 1100

#### Emisor en C

```
PS C:\Users\Personal\Documents\Universidad\8 Semestre\Redes\Laboratorio2-Redes\Hamming> ./Emisor
Ingrese los bits de datos: 1100
Bits de paridad: 0 0 1
Codigo Hamming: 1100001
```

#### Receptor en Python:

```
*****
Bienvenido al algoritmo de Hamming (Receptor)
*****
Ingresa los bits de datos recibidos del emisor: 1100001
Bloque 1 (no hubo error): [1, 1, 0, 0, 0, 0, 1]
El mensaje es: 1100
```

Mensaje 2: 10110

#### Emisor en C

```
PS C:\Users\Personal\Documents\Universidad\8 Semestre\Redes\Laboratorio2-Redes\Hamming> ./Emisor
Ingrese los bits de datos: 10110
Bits de paridad: 0 1 1
Bits de paridad: 0 1 1
Codigo Hamming: 01100110000111
```

#### Receptor en Python:

```
*****
Bienvenido al algoritmo de Hamming (Receptor)
*****
Ingresa los bits de datos recibidos del emisor: 01100110000111
Bloque 1 (no hubo error): [0, 1, 1, 0, 0, 1, 1]
Bloque 2 (no hubo error): [0, 0, 0, 0, 1, 1, 1]
El mensaje es: 0110001
```

Mensaje 3: 10101010

#### Emisor en C

```
PS C:\Users\Personal\Documents\Universidad\8 Semestre\Redes\Laboratorio2-Redes\Hamming> ./Emisor
Ingrese los bits de datos: 10101010
Bits de paridad: 0 1 0
Bits de paridad: 0 1 0
Codigo Hamming: 10100101010010
```

#### Receptor en Python:

```
*****
Bienvenido al algoritmo de Hamming (Receptor)
*****
Ingresa los bits de datos recibidos del emisor: 10100101010010
Bloque 1 (no hubo error): [1, 0, 1, 0, 0, 1, 0]
Bloque 2 (no hubo error): [1, 0, 1, 0, 0, 1, 0]
El mensaje es: 10101010
```

## Algoritmo CRC-32:

### Mensaje 1: 1100

Emisor en Python

```
└─$ cd ~/Documents/OctavoSemestre/Redes/Laboratorios/Laboratorio 2/CRCAlgorithm/Emisor & python main !2 ..... ⏺ 04:45:15 PM
> /opt/homebrew/bin/python3 "/Users/estebandonis/Documents/OctavoSemestre/Redes/Laboratorios/Laboratorio 2/CRCAlgorithm/Emisor/emisor.py"
Ingrese el mensaje: 1100
Mensaje Extendido: 110000110001110011011000011011010011
```

Receptor en C#

```
└─$ cd ~/Doc/0/Redes/Laboratorios/Laboratorio 2/CRCAlgorithm/Receptor & python main !9 ..... ⏺ 04:45:54 PM
> /Users/estebandonis/.vscode/extensions/ms-dotnettools.csharp-2.39.29-darwin-arm64/.debugger/arm64/vsdbg --interpreter=vscode --connection=/var/folders/nh/qxcg2qs54dq8v4tnbxqzpm7w0000gn/T/CoreFxPipe_vsdbg-ui-84ce2e81ba1e4c418cd4e39dba0bfd6d
Ingrese el mensaje: 110000110001110011011000011011010011
El mensaje recibido: 1100 no contiene errores.
```

### Mensaje 2: 10110

Emisor en Python

```
└─$ cd ~/Documents/OctavoSemestre/Redes/Laboratorios/Laboratorio 2/CRCAlgorithm/Emisor & python main !9 ..... ⏺ 04:45:54 PM
> /opt/homebrew/bin/python3 "/Users/estebandonis/Documents/OctavoSemestre/Redes/Laboratorios/Laboratorio 2/CRCAlgorithm/Emisor/emisor.py"
Ingrese el mensaje: 10110
Mensaje Extendido: 1011001010010010101101000101101110101
```

Receptor en C#

```
└─$ cd ~/Doc/0/Redes/Laboratorios/Laboratorio 2/CRCAlgorithm/Receptor & python main !9 ..... ⏺ 04:45:54 PM
> /Users/estebandonis/.vscode/extensions/ms-dotnettools.csharp-2.39.29-darwin-arm64/vsdbg --interpreter=vscode --connection=/var/folders/nh/qxcg2qs54dq8v4tnbxqzpm7w0000gn/T/CoreFxPipe_vsdbg-ui-31ba4c4b2e2e47afb0d6ce97125403fc
Ingrese el mensaje: 1011001010010010101101000101101110101
El mensaje recibido: 10110 no contiene errores.
```

### Mensaje 3: 10101010

Emisor en Python

```
└─$ cd ~/Documents/OctavoSemestre/Redes/Laboratorios/Laboratorio 2/CRCAlgorithm/Emisor & python main !9 ..... ⏺ 04:45:54 PM
> /opt/homebrew/bin/python3 "/Users/estebandonis/Documents/OctavoSemestre/Redes/Laboratorios/Laboratorio 2/CRCAlgorithm/Emisor/emisor.py"
Ingrese el mensaje: 10101010
Mensaje Extendido: 1010101011011010011001001001110101101111
```

Receptor en C#

```
~/Doc/OctavoSemestre/Redes/Laboratorios/Laboratorio 2/CRCAlgorithm/Receptor
~/Users/estebandonis/.vscode/extensions/ms-dotnettools.csharp-2.39.29-darwin-arm
g2qs54dq8v4tnbxqzpm7w0000gn/T/CoreFxPipe_vsdbg-ui-79250b8368064f38b560e5628f4960df
Ingrese el mensaje: 10101011011010011001001110101101111
El mensaje recibido: 10101010 no contiene errores.
```

- Enviar un mensaje al emisor, copiar el mensaje generado por este y cambiar un bit cualquiera antes de proporcionar 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 Hamming(7,4):

Mensaje 1: **10101**

Emisor en C: **01011010000111**

```
PS C:\Users\Personal\Documents\Universidad\8 Semestre\Redes\Laboratorio2-Redes\Hamming> ./Emisor
Ingrese los bits de datos: 10101
Bits de paridad: 1 0 1
Bits de paridad: 0 1 1
Codigo Hamming: 01011010000111
```

Receptor en Python:**01011010001111**

```
*****
Bienvenido al algoritmo de Hamming (Receptor)
*****
Ingresa los bits de datos recibidos del emisor: 01011010001111
Bloque 1 (no hubo error): [0, 1, 0, 1, 1, 0, 1]
Bloque 2 (error corregido en la posición 4): [0, 0, 0, 0, 1, 1, 1]
Mensaje corregido: [0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1]
```

Mensaje 2: **0101**

Emisor en C:**0101101**

```
PS C:\Users\Personal\Documents\Universidad\8 Semestre\Redes\Laboratorio2-Redes\Hamming> ./Emisor
Ingrese los bits de datos: 0101
Bits de paridad: 1 0 1
Codigo Hamming: 0101101
```

Receptor en Python:**1101101**

```
*****
Bienvenido al algoritmo de Hamming (Receptor)
*****
Ingresa los bits de datos recibidos del emisor: 1101101
Bloque 1 (error corregido en la posición 7): [0, 1, 0, 1, 1, 0, 1]
Mensaje corregido: [0, 1, 0, 1, 1, 0, 1]
```

Mensaje 3: **11110000**

Emisor en C: **00000001111111**

```
PS C:\Users\Personal\Documents\Universidad\8 Semestre\Redes\Laboratorio2-Redes\Hamming> ./Emisor
Ingrese los bits de datos: 11110000
Bits de paridad: 0 0 0
Bits de paridad: 1 1 1
Codigo Hamming: 00000001111111
```

Receptor en Python: **00100001111111**

```
*****
Bienvenido al algoritmo de Hamming (Receptor)
*****
Ingresa los bits de datos recibidos del emisor: 00100001111111
Bloque 1 (error corregido en la posición 5): [0, 0, 0, 0, 0, 0, 0]
Bloque 2 (no hubo error): [1, 1, 1, 1, 1, 1, 1]
Mensaje corregido: [0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1]
```

Algoritmo CRC-32:

Mensaje 1: **10101**

Emisor en Python: 1010101011011101010010110000000011011

```
└─$ ➜ ~/Documents/OctavoSemestre/Redes/Laboratorios/Laboratorio 2/CRCAlgorithm/Emisor ➜ main !9 ⏎ 5s ⏎ 04:49:29 PM
  > /opt/homebrew/bin/python3 "/Users/estebandonis/Documents/OctavoSemestre/Redes/Laboratorios/Laboratorio 2/CRCAlgorithm/Emisor/emisor.py"
Ingrese el mensaje: 10101
Mensaje Extendido: 1010101011011101010010110000000011011
```

Receptor en C#: 101010101**0**011110101001011000000011011

```
└─$ ➜ ~/Doc/0/Redes/Laboratorios/Laboratorio 2/CRCAlgorithm/Receptor ➜ main !9 ⏎ ... x INT ⏎ 04:53:44 PM
  > /Users/estebandonis/.vscode/extensions/ms-dotnettools.csharp-2.39.29-darwin-arm64/.debugger/arm64/vsdbg
    --interpreter=vscode --connection=/var/folders/nh/qxcg2qs54dq8v4tnbxqzpm7w0000gn/T/CoreFxPipe_vsdbg-ui-40
4d56ed032641c685b48717d4250378
Ingrese el mensaje: 1010101010011110101001011000000011011
Error: Se descarta el mensaje por contener errores.
```

Mensaje 2: **0101**

Emisor en Python: 01010001011110001010110101101101011

```
└─$ ➜ ~/Documents/OctavoSemestre/Redes/Laboratorios/Laboratorio 2/
  > /opt/homebrew/bin/python3 "/Users/estebandonis/Documents/OctavoSemestre/Redes/Laboratorios/Laboratorio 2/CRCAlgorithm/Receptor" ➜ main !9 ⏎ 5s ⏎ 04:53:44 PM
Ingrese el mensaje: 0101
Mensaje Extendido: 01010001011110001010110101101101011
```

Receptor en C#: 0101000**0**011110001010110101101101011

```
└─$ ➜ ~/Doc/OctavoSemestre/Redes/Laboratorios/Laboratorio 2/CRCAlgorithm/Receptor ➜ main !9 ⏎ 5s ⏎ 04:53:44 PM
  > /Users/estebandonis/.vscode/extensions/ms-dotnettools.csharp-2.39.29-darwin-arm64/g2qs54dq8v4tnbxqzpm7w0000gn/T/CoreFxPipe_vsdbg-ui-0b884117a925450db9bdb84c9966e540
Ingrese el mensaje: 01010000111110001010110101101101011
Error: Se descarta el mensaje por contener errores.
```

Mensaje 3: **11110000**

Emisor en Python: 111100001000100110111000111110100001001

```
apple ~ ~/Documents/OctavoSemestre/Redes/Laboratorios/Laboratorio 2/0
> /opt/homebrew/bin/python3 "/Users/estebandonis/Documents/OctavoS
Ingrese el mensaje: 11110000
Mensaje Extendido: 1111000010001001101110001111110100001001
```

Receptor en C#: 1111000010001101101110001111110100001001

```
apple ~ ~/Doc/OctavoSemestre/Redes/Laboratorios/Laboratorio 2/CRAAlgorithm/Receptor
> /Users/estebandonis/.vscode/extensions/ms-dotnettools.csharp-2.39.29-darwin-arm
g2qs54dq8v4tnbxqzpm7w0000gn/T/CoreFxPipe_vsdbg-ui-159987c5b3f04b55801180242d0d0927
Ingrese el mensaje: 1111000010001101101110001111110100001001
Error: Se descarta el mensaje por contener errores.
```

- Enviar un mensaje al emisor, copiar el mensaje generado por este y cambiar dos o más bits cualesquiera antes de proporcionar 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 Hamming(7,4):

Mensaje 1: 0001101

Emisor en C:1100110000000

```
PS C:\Users\Personal\Documents\Universidad\8 Semestre\Redes\Laboratorio2-Redes\Hamming> ./Emisor
Ingrese los bits de datos: 0001101
Bits de paridad: 0 1 0
Bits de paridad: 0 0 0
Codigo Hamming: 1100110000000
```

Receptor en Python:11001100011000

```
*****
Bienvenido al algoritmo de Hamming (Receptor)
*****
Ingresa los bits de datos recibidos del emisor: 11001100011000
Bloque 1 (no hubo error): [1, 1, 0, 0, 1, 1, 0]
Bloque 2 (error corregido en la posición 1): [0, 0, 1, 1, 0, 0, 1]
Mensaje corregido: [1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1]
```

Mensaje 2: 10010110

Emisor en C:01100111001100

```
PS C:\Users\Personal\Documents\Universidad\8 Semestre\Redes\Laboratorio2-Redes\Hamming> ./Emisor
Ingrese los bits de datos: 10010110
Bits de paridad: 0 1 1
Bits de paridad: 1 0 0
Codigo Hamming: 01100111001100
```

Receptor en Python:00000111000000

```
*****
Bienvenido al algoritmo de Hamming (Receptor)
*****
Ingresa los bits de datos recibidos del emisor: 00000111000000
Bloque 1 (error corregido en la posición 3): [0, 0, 0, 0, 1, 1, 1]
Bloque 2 (error corregido en la posición 7): [0, 0, 0, 0, 0, 0, 0]
Mensaje corregido: [0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0]
```

**Mensaje 3: 11101**

Emisor en C:**1100110000011**

```
PS C:\Users\Personal\Documents\Universidad\8 Semestre\Redes\Laboratorio2-Redes\Hamming> ./Emisor
Ingrese los bits de datos: 11101
Bits de paridad: 0 1 0
Bits de paridad: 0 1 1
Codigo Hamming: 11001100000111
```

Receptor en Python:**10100100010010**

```
*****
Bienvenido al algoritmo de Hamming (Receptor)
*****
Ingresa los bits de datos recibidos del emisor: 10100100010010
Bloque 1 (no hubo error): [1, 0, 1, 0, 0, 1, 0]
Bloque 2 (error corregido en la posición 7): [1, 0, 1, 0, 0, 1, 0]
Mensaje corregido: [1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0]
```

**Algoritmo CRC-32:**

**Mensaje 1: 0001101**

Emisor en Python: 000110100110001110011011000011011010011

```
└─ Apple ➜ ~/Documents/OctavoSemestre/Redes/Laboratorios/Laboratorio 2/CRCAlgorithm/Emisor ✘ P main !9 ..... ⚡ 05:00:19 PM
  └─ /opt/homebrew/bin/python3 "/Users/estebandonis/Documents/OctavoSemestre/Redes/Laboratorios/Laboratorio 2/CRCAlgorithm/Emisor/emisor.py"
    Ingrese el mensaje: 0001101
    Mensaje Extendido: 000110100110001110011011000011011010011
```

Receptor en C#: **001110100110001010011011000010011010011**

```
└─ Apple ➜ ~/Doc/0/Redes/Laboratorios/Laboratorio 2/CRCAlgorithm/Receptor ✘ P main !9 ..... x INT ⚡ 05:09:22 PM
  └─ /Users/estebandonis/.vscode/extensions/ms-dotnettools.csharp-2.39.29-darwin-arm64/.debugger/arm64/vsdbg
    --interpreter=vscode --connection=/var/folders/nh/qxcg2qs54dq8v4tnbxqzpm7w0000gn/T/CoreFxPipe_vsdbg-ui-2f
    846d5c2a4440008b2af100cd001f8b
    Ingrese el mensaje: 001110100110001010011011000010011010011
    Error: Se descarta el mensaje por contener errores.
```

**Mensaje 2: 10010110**

Emisor en Python: 100101100011101101011010011010110011011

```
└─ Apple ➜ ~/Documents/OctavoSemestre/Redes/Laboratorios/Laboratorio 2/CRCAlgorithm/Emisor ✘ P main !9 ..... x INT ⚡ 05:09:22 PM
  └─ /opt/homebrew/bin/python3 "/Users/estebandonis/Documents/OctavoSemestre/Redes/Laboratorios/Laboratorio 2/CRCAlgorithm/Emisor/emisor.py"
    Ingrese el mensaje: 10010110
    Mensaje Extendido: 100101100011101101011010011010110011011
```

Receptor en C#: **000101110011101101001010010011011**

```
~/Doc/OctavoSemestre/Redes/Laboratorios/Laboratorio 2/CRCAlgorithm/Receptor
> /Users/estebandonis/.vscode/extensions/ms-dotnettools.csharp-2.39.29-darwin-arm
g2qs54dq8v4tnbxqzpm7w000gn/T/CoreFxPipe_vsdbg-ui-eec3691bb4b4759b98a93e50268c986
Ingrese el mensaje: 0001011100111011010010100100101100011011
Error: Se descarta el mensaje por contener errores.
```

### Mensaje 3: 11101

Emisor en Python: 1110101111011101100010110110100011

```
~/Documents/OctavoSemestre/Redes/Laboratorios/Laboratorio 2/
> /opt/homebrew/bin/python3 "/Users/estebandonis/Documents/Octavo
Ingrese el mensaje: 11101
Mensaje Extendido: 111010111101110111000101110110100011
```

Receptor en C#: 111100111011001110010011110100010

```
~/Doc/OctavoSemestre/Redes/Laboratorios/Laboratorio 2/CRCAlgorithm/Receptor
> /Users/estebandonis/.vscode/extensions/ms-dotnettools.csharp-2.39.29-darwin-arm
g2qs54dq8v4tnbxqzpm7w000gn/T/CoreFxPipe_vsdbg-ui-1cccd903e6fa64363a2035d859525570e
Ingrese el mensaje: 111100111011001110010011110100010
Error: Se descarta el mensaje por contener errores.
```

- ¿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.

### Algoritmo Hamming(7,4):

R// Si es posible, porque el algoritmo Hamming solo es posible que detecte 1 error y corregirlo, lo cual si se pasa 2 errores este confunde los bits de paridad el cual hace que en algunos casos diga que hay un error en alguna posición el cual ni fue la que se cambiaron los bits en otros casos al cambiar más de dos bits puede que llegue a un punto donde no le marque que hubo error lo cual es una desventaja de este algoritmo ya que el no sabe si hubo más de 2 cambios, por lo cual a continuación se les mostrará un mensaje sencillo el cual se le van a alterar los bits de tal manera que el resultado sea que no hubo error.

### Ejemplo Hamming(7,4):

**Mensaje: 1101**

**Emisor C:1100110**

```
C:\Users\Personal\Documents\Universidad\8 Semestre\Redes\Laboratorio2-Redes\Hamming> ./Emisor
Ingrese los bits de datos: 1101
Bits de paridad: 0 1 0
Codigo Hamming: 1100110
```

**Receptor python:1010010**

```
*****
Bienvenido al algoritmo de Hamming (Receptor)
*****
Ingresa los bits de datos recibidos del emisor: 1010010
Bloque 1 (no hubo error): [1, 0, 1, 0, 0, 1, 0]
El mensaje es: 1010
PS C:\Users\Personal\Documents\Universidad\8 Semestre\Redes\Laboratorio2-Redes\Hamming> █
```

Como podemos observar es posible manipular bits para que no detecte error lo cual podemos ver que no detectó ningún error lo cual hace que el mensaje se disuelva o bien no llegue como se esperaba ya que si se dan cuenta el mensaje es 1101 mientras que el resultado es 1010 lo cual es algo que nada que ver con el mensaje original.

### Algoritmo CRC-32:

R// Si es posible, sin embargo, depende mucho del grado del algoritmo del polinomio. Como sabemos, ningún algoritmo es perfecto, por lo que cada uno tiene sus limitaciones al igual que el CRC. Por lo que la fortaleza del algoritmo recae más que todo en el grado del polinomio, ya que, como podemos ver en el ejemplo siguiente, el CRC-3 no es tan robusto que solo le podemos cambiar el primero y último bit para que no detecte el error. Pero cuando probamos lo mismo con CRC-32, podemos observar que no funciona, esto no significa que es imposible, pero a por tener una verificación de un polinomio de 32 significa que es menos probable encontrar la forma de quebrarlo.

#### CRC - 3: 1010

Emisor en Python: 1010011

```
apple ➜ ~/Documents/OctavoSemestre/
apple ➜ /opt/homebrew/bin/python3 "/U
Ingrrese el mensaje: 1010
Mensaje Extendido: 1010011
```

Receptor en C#: 0010010

```
apple ➜ ~/Doc/OctavoSemestre/Redes/Laboratorios/Laboratorio 2/CRCAlgori
apple ➜ /Users/estebandonis/.vscode/extensions/ms-dotnettools.csharp-2.39.
xqzpm7w0000gn/T/CoreFxPipe_vsdbg-ui-bcbfd8209d854c17950b96be131d1719
Ingrrese el mensaje: 0010010
El mensaje recibido: 0010 no contiene errores.
```

#### CRC - 32: 1010

Emisor en Python: 101000101011010010111100101101100001

```
apple ➜ ~/Documents/OctavoSemestre/Redes/Laboratorios/Laboratorio 2/
apple ➜ /opt/homebrew/bin/python3 "/Users/estebandonis/Documents/Octavo
Ingrrese el mensaje: 1010
Mensaje Extendido: 101000101011010010111100101101100001
```

Receptor en C#: 001000101011010010111100101101100000

```
apple ➜ ~/Doc/OctavoSemestre/Redes/Laboratorios/Laboratorio 2/CRCAlgoritmo
↳ /Users/estebandonis/.vscode/extensions/ms-dotnettools.csharp-2.39.
xqzpm7w0000gn/T/CoreFxPipe_vsdbg-ui-6833bd5baad84488a5b1fda5cda59f51
Ingrese el mensaje: 001000101011010010111100101101100000
Error: Se descarta el mensaje por contener errores.
```

- 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, al modificar dos bits pude observar que en algunos casos el algoritmo no era capaz de detectar el error, esto es una desventaja con respecto a los otros métodos. Uno de estos casos fue el mensaje 1111 el cual cambió a 1100 pero el valor del bit de paridad fue el mismo.”

### Algoritmo Hamming(7,4):

R//Al implementar el algoritmo hamming(7,4) observe varias ventajas y desventajas de este algoritmo. Las ventajas del algoritmo hamming hay varias una de ellas sería la redundancia ya que como nos mostraron en la clase se añaden solo 3 bits de paridad y 4 bits de datos lo que nos resulta un total de 7 bits lo cual siempre nos da valores de 7 bits, otra es la corrección de errores ya que hamming es capaz de corregir un error lo cual nos sirve cuando se espera una tasa de errores bajas otra ventaja es la facilidad de implementar ya que sus pasos son sencillos y relativamente fáciles de resolver y por último la velocidad de ejecución al ser relativamente sencillo y corto podemos ver que la velocidad de respuesta es mucho más óptimo a comparación con el algoritmo CRC. Ahora para las desventajas como bien vimos en las pruebas la limitación de correcciones ya que este algoritmo solo puede corregir 1 error lo cual no podrá corregir el mensaje correctamente, otra desventaja sería la tasa de detección de errores como bien podemos observar en las pruebas vemos que no es capaz de detectar más de 2 errores e incluso en varias ocasiones detectar 2 le cuesta bastante ya que a veces tiende a confundirse con los bits de paridad por estas razones este algoritmo no es muy bueno que digamos. Comparando ambos algoritmos podemos ver que hamming es menos redundante debido a que son 3 bits mientras que CRC-32 son 32 bits, además hamming puede corregir solo 1 bit de error mientras que CRC-32 no tiene la capacidad para corregir errores y por último la complejidad y velocidad, hamming es muchos más sencillo de implementar y la velocidad es mucho más rápida debido a su sencilla implementación ya que no posee muchos bits, mientras que CRC-32 es mucho más lento debido a sus 32 bits.

### **Algoritmo CRC-32:**

R// En la implementación del algoritmo CRC-32, pude notar que fue bastante fácil de realizar en un programa, las operaciones e instrucciones que se realizan en el proceso son bastante simples y redundantes en el emisor y el receptor. También pude ver que a medida que utilizamos un polinomio con mayor grado, más seguros podemos estar de que nuestro algoritmo será capaz de identificar cualquier error, como se pudo ver al momento de enviar el mensaje 1010, y cambiar el primero y el último bit. Esto puede crear más dificultad a la hora de enviar el mensaje, porque tenemos un mensaje mucho más largo comparado con Hamming 7,4 y una complejidad computacional mucho más alta, al tener que realizar la operación XOR muchas más veces; sin embargo, al ser una operación básica, está no representa mucha carga para la unidad de procesamiento. Considero que es mucho más fácil al momento de manejar mensajes más largos, ya que, a diferencia de Hamming 7, 4, nuestros mensajes de verificación no poseen una cantidad determinada, sino que podemos enviar, dentro del mismo mensaje, la cantidad de bits que queramos sin vernos limitados. Algo notable es que puede determinar que el mensaje tiene error, incluso cambiando más de 4 o 5 bits, lo cual, no es algo que Hamming 7, 4 pueda hacer con tanta facilidad como se observó en el ejemplo.