

# Laboratorio 2 – Esquemas de detección y corrección de errores (Parte 1)

---

## Integrantes

Carlos Valladares – Carné 221164

Brandon Reyes – Carné 22992

## 1. Descripción de la práctica

En este laboratorio se abordan los mecanismos de detección y corrección de errores en comunicaciones, enfocándonos en la capa de Enlace. Se implementan y prueban al menos dos algoritmos: uno de corrección (Hamming (7,4)) y otro de detección.

### Objetivos:

1. Analizar el funcionamiento de algoritmos de detección y corrección.
2. Implementar emisor y receptor en diferentes lenguajes.
3. Comparar ventajas y desventajas de cada algoritmo.

## 2. Implementación

### 2.1 Algoritmo Hamming (7,4)

#### Emisor (Rust):

```
use std::fs::File;
use std::io::{self, Write};

fn main() {
    println!("Ingrese el mensaje en binario (ejemplo: 1011001):");

    let mut entrada = String::new();
    io::stdin().read_line(&mut entrada).expect("Error al leer entrada");
    let entrada = entrada.trim();

    if !entrada.chars().all(|c| c == '0' || c == '1') {
        println!("Error: el mensaje debe contener solo 0s y 1s.");
        return;
    }
}
```

```

let mut bits: Vec<u8> = entrada.chars().map(|c| c.to_digit(2).unwrap() as u8).collect();

while bits.len() % 4 != 0 {
    bits.push(0);
}

let mut trama_codificada = Vec::new();

for bloque in bits.chunks(4) {
    let d = bloque;
    let p1 = d[0] ^ d[1] ^ d[3];
    let p2 = d[0] ^ d[2] ^ d[3];
    let p3 = d[1] ^ d[2] ^ d[3];

    // Orden estándar: P1 P2 D1 P3 D2 D3 D4
    trama_codificada.push(p1);
    trama_codificada.push(p2);
    trama_codificada.push(d[0]);
    trama_codificada.push(p3);
    trama_codificada.push(d[1]);
    trama_codificada.push(d[2]);
    trama_codificada.push(d[3]);
}

let salida: String = trama_codificada.iter().map(|b| b.to_string()).collect();

// Mostrar en pantalla
println!("Mensaje codificado en Hamming (7,4): {}", salida);

// Guardar en archivo
let mut archivo = File::create("mensaje_codificado.txt").expect("No se pudo crear el
archivo");
writeln!(archivo, "Mensaje codificado en Hamming (7,4): {}", salida)
    .expect("No se pudo escribir en el archivo");
}

```

### **Receptor (Python):**

```

# Matriz H (3×7) para comprobar las 3 sumas de paridad
H_MATRIX = [
    [1, 0, 1, 0, 1, 0, 1], # chequeo de P1 sobre bits [0,2,4,6]
    [0, 1, 1, 0, 0, 1, 1], # chequeo de P2 sobre bits [1,2,5,6]
    [0, 0, 0, 1, 1, 1, 1], # chequeo de P3 sobre bits [3,4,5,6]

```

]

```
def receiver_hamming74(codeword: str) -> str:
    recovered = []
    total_blocks = len(codeword) // 7

    for blk_idx in range(total_blocks):
        # Extraemos bloque de 7 bits como lista de enteros
        start = blk_idx * 7
        block = list(map(int, codeword[start:start+7]))

        # Calcular multiplicando  $H\_MATRIX \times block \pmod{2}$ 
        syndrome = [
            sum(h_bit * block[j] for j, h_bit in enumerate(row)) % 2
            for row in H_MATRIX
        ]
        # Conversión de bits de síndrome a posición de error (1..7)
        err_pos = syndrome[0] + 2*syndrome[1] + 4*syndrome[2]

        print(f"== Bloque #{blk_idx+1} ==")
        if err_pos == 0:
            print("No se detectó error.")
        else:
            print(f"Error en posición {err_pos}, corrigiendo bit ...")
            # Corregimos invirtiendo el bit
            block[err_pos-1] ^= 1

        # Índices donde están los bits de datos D1,D2,D3,D4
        data_indices = [2, 4, 5, 6]
        data_bits = [block[i] for i in data_indices]

        print("Bloque corregido: ", ".join(str(b) for b in block)")
        print("Datos extraídos: ", ".join(str(b) for b in data_bits)")
        print()

        recovered.append(".join(str(b) for b in data_bits))

    return ".join(recovered)

if __name__ == "__main__":
    raw = input("Introduce la cadena Hamming (7,4): ").strip()
    # Validación básica
```

```

if not raw or any(ch not in "01" for ch in raw):
    print("¡ERROR! Solo se admiten caracteres '0' y '1'.")
    exit(1)

# Asegurar longitud múltiplo de 7 añadiendo padding al final
rem = len(raw) % 7
if rem != 0:
    extra = 7 - rem
    print(f"Longitud no múltiplo de 7: añadido {extra} ceros de relleno.")
    raw += "0" * extra

resultado = receiver_hamming74(raw)
print("Mensaje final recompuesto:", resultado)

```

### 3. Escenarios de prueba y resultados

Se realizaron los siguientes casos de prueba para Hamming (7,4):

- Sin errores
- Un error en un bit
- Dos o más errores en el mismo bloque

Para cada caso se documentaron la trama original, la trama codificada, los bits alterados y la salida del receptor.

Ejecución del emisor

```

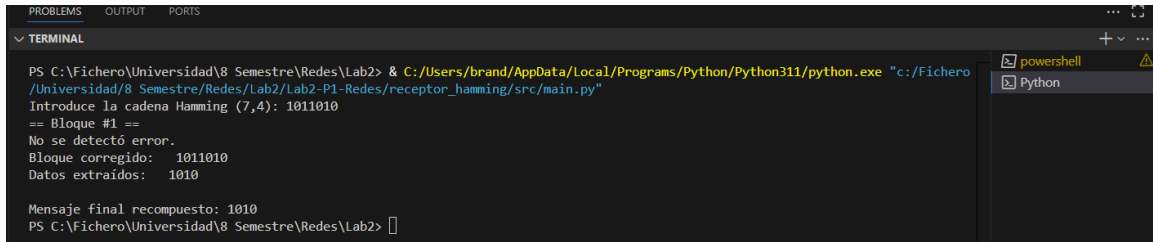
▼ TERMINAL

d-----      23/07/2025      11:55              target
-a-----      23/07/2025      11:55          158 Cargo.lock
-a-----      23/07/2025      11:52           85 Cargo.toml
-a-----      23/07/2025      12:35           45 mensaje_codificado.txt

PS C:\Fichero\Universidad\8 Semestre\Redes\Lab2\Lab2-P1-Redes\emisor_hamming> cargo run
Finished `dev` profile [unoptimized + debuginfo] target(s) in 0.18s
Running `target\debug\emisor_hamming.exe`
Ingrese el mensaje en binario (ejemplo: 1011001):
1010
Mensaje codificado en Hamming (7,4): 1011010
PS C:\Fichero\Universidad\8 Semestre\Redes\Lab2\Lab2-P1-Redes\emisor_hamming>

```

Ejecución del receptor



```
PROBLEMS OUTPUT PORTS
TERMINAL
PS C:\Fichero\Universidad\8 Semestre\Redes\Lab2> & C:/Users/brand/AppData/Local/Programs/Python/Python311/python.exe "c:/Fichero
/Universidad/8 Semestre/Redes/Lab2/Lab2-P1-Redes/receptor_hamming/src/main.py"
Introduce la cadena Hamming (7,4): 1011010
== Bloque #1 ==
No se detectó error.
Bloque corregido: 1011010
Datos extraídos: 1010

Mensaje final recompueto: 1010
PS C:\Fichero\Universidad\8 Semestre\Redes\Lab2> 
```

## 4. Análisis de detección fallida

Hamming (7,4) corrige un único bit y detecta hasta dos errores. Cuando hay dos errores el síndrome apunta a una posición concreta, pero la corrección no restaura el mensaje original, mostrando la limitación del algoritmo.

## 5. Ventajas y desventajas

- Complejidad: baja, cálculos XOR rápidos.
- Overhead: 3 bits de paridad por cada 4 de datos.
- Velocidad: muy alta (operaciones bit a bit).
- Capacidad: corrige 1 bit, detecta 2; no corrige 2.
- Robustez: suficiente para enlaces con baja tasa de errores.

## 6. Conclusiones

La implementación de Hamming (7,4) demuestra un equilibrio entre eficiencia y capacidad de corrección de un solo bit. Para entornos con mayor tasa de errores, se recomienda usar códigos más robustos o combinaciones con checksum/CRC.

## Apéndice: Código complete

Enlace de GitHub: <https://github.com/BrandonReyes0609/Lab2-P1-Redes>