Desafío 1 — Reconstrucción de mensaje comprimido y encriptado

**Curso:** Informática II

**Autores:** Carlos Mario Rendón Martínez y Michell

**Profesor:** Aníbal Guerra / Augusto Salazar

**Semestre: 2025-2**

**Desafio-1.**

1. Contextualización
El desafío consiste en reconstruir un mensaje original que fue comprimido (usando RLE o LZ78) y luego encriptado (con rotación de bits y XOR).
El único dato que se entrega es el mensaje final encriptado (Encriptado.txt) y un fragmento conocido del texto original (pistaX.txt).

La tarea fue:

1) Identificar qué método de compresión se utilizó.
2)Descubrir los parámetros de encriptación: número de bits de rotación (n) y clave XOR (K).
3)Desencriptar el mensaje.
4)Descomprimirlo para recuperar el texto original.

Restricciones:

-Usar **C++ con Qt**.
-No se permite usar `string` ni STL.
-Obligatorio usar punteros, arreglos y memoria dinámica.

2. Análisis del problema

Las principales dificultades que se nos presentaron durante el transcurso del desafío fueron las siguientes:

-No se especificaba qué compresión se había aplicado (RLE o LZ78).
-El número de bits rotados (n) podía estar entre 1 y 7.
-La clave XOR (K) podía ser cualquier valor de 1 byte (0–255).

Durante cada paso que dábamos con el desarrollo del programa llegamos a la conclusión de realizar pruebas exhaustivas combinando métodos de compresión con múltiples parámetros de encriptación hasta encontrar un resultado válido que contuviera la pista.

Observaciones iniciales:

-En Encriptado1.txt y Encriptado2.txt se repetía el patrón Z, lo que sugería relación con la clave 0x5A (ASCII de 'Z').

-En los casos 3 y 4 se veían patrones más ordenados (`aaaaa`, `bbbbb`, `cccccc`), compatibles con RLE y LZ78.

-Las pistas (`rrenosdes`, `xyyyyzzzza`) confirmaban que la descompresión debía recuperar texto coherente.

3. Diseño de la solución

La solución se dividió en tres módulos principales:

1) Encriptación / Desencriptación
Archivos: encriptacion.cpp, encriptacion.h
Funciones:

- rotar_izquierda y rotar_derecha: rotación de bits con operaciones de desplazamiento.
- aplicar_clave: aplica XOR con la clave K.
- procesar_mensaje: recorre el mensaje aplicando primero XOR y luego rotación (o al revés, en modo encriptar).

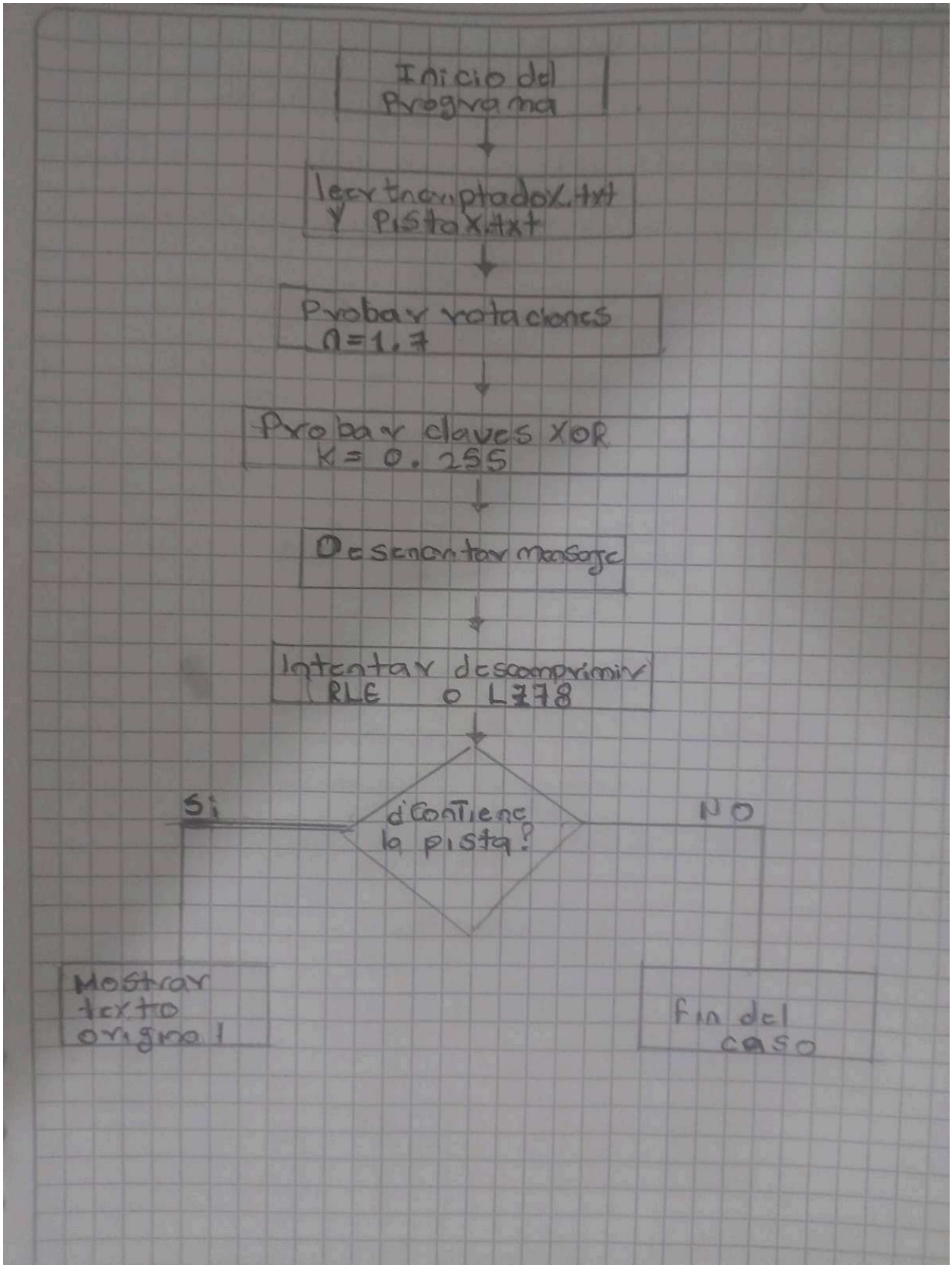
2) Compresión / Descompresión
Archivos: compresion.cpp, compresion.h
Funciones:

- descomprimir_rle: expansión de secuencias numéricas.
- descomprimir_rle_binario y descomprimir_rle_binario16: variantes binarias.
- convertirLZ78: interpreta tripletas índice-letra.
- descomprimir_lz78: reconstruye el texto usando diccionarios dinámicos.

c) Programa principal
Archivo: main.cpp
Responsabilidades:

- Preguntar cuántos casos se evaluarán.
- Leer EncriptadoX.txt y pistaX.txt.
- Probar todas las combinaciones de rotaciones (1..7) y claves candidatas.
- Intentar descomprimir con RLE o LZ78.
- Verificar si la pista aparece en el resultado.

Diagrama de flujo de la solución:


Inicio del Programa
leer tncriptadoX.txt y PistaX.txt
Probar rotaciones n=1..7
Probar claves XOR K=0..255
Desencriptar mensaje
Intentar descomprimir RLE o LZ78
¿Contiene la pista?
Si
NO
Mostrar texto original
Fin del caso

4. Implementación (fragmentos clave)

Lectura de archivos

```
char* leerArchivo(const char* nombreArchivo, int& largo) {
    FILE* archivo = fopen(nombreArchivo, "rb");
    if (!archivo) return NULL;
    fseek(archivo, 0, SEEK_END);
    long t = ftell(archivo);
    largo = (int)t;
    rewind(archivo);
    char* contenido = new char[largo + 1];
    fread(contenido, 1, largo, archivo);
    contenido[largo] = '\0';
    fclose(archivo);
    return contenido;
}
```

Rotación y XOR

```
uint8_t rotar_derecha(uint8_t byte, unsigned int pasos) {
    pasos = pasos % 8;
    uint16_t temp = byte;
    return (uint8_t)(((temp >> pasos) | (temp << (8 - pasos))) & 0xFF);
}

uint8_t aplicar_clave(uint8_t byte, uint8_t clave) {
    return byte ^ clave;
}

```

Descompresión LZ78

```
char* descomprimir_lz78(Datos* arreglo, int cantidad) {
    char** diccionario = new char*[cantidad + 1];
    diccionario[0] = new char[1]{'\0'};
    int total = 0;
    for (int i = 0; i < cantidad; i++) {
        int idx = arreglo[i].numero;
        char letra = arreglo[i].letra;
        int largo_prefijo = strlen(diccionario[idx]);
        char* palabra = new char[largo_prefijo + 2];
        memcpy(palabra, diccionario[idx], largo_prefijo);
        palabra[largo_prefijo] = letra;
        palabra[largo_prefijo + 1] = '\0';
        diccionario[i + 1] = palabra;
        total += largo_prefijo + 1;
    }
    // reconstrucción final…
}
```

5. Pruebas y resultados

Según el archivo README.txt dado por el profe, los parámetros correctos para cada caso fueron:

- Caso 1 – Encriptado1.txt
    - Compresión: RLE
    - Rotación: 3
    - Clave: 0x5A
    - Texto recuperado: descripción larga de paisajes naturales (incluye “terrenosdesconocidos”).
- Caso 2 – Encriptado2.txt
    - Compresión: LZ78
    - Rotación: 3
    - Clave: 0x5A

    - Texto recuperado: relatos de exploradores y misterios históricos.

- Caso 3 – Encriptado3.txt
    - Compresión: RLE
    - Rotación: 3
    - Clave: 0x40
    - Texto recuperado: secuencias de letras repetidas (aaaaa…zzzz).
- Caso 4 – Encriptado4.txt
    - Compresión: LZ78
    - Rotación: 3
    - Clave: 0x40
    - Texto recuperado: secuencias similares de letras repetidas.

El programa siempre mostró la validación en consola:
Metodo: RLE (binario16) o Metodo: LZ78 seguido del texto final.

6. Problemas encontrados

- Lectura de archivos: al inicio no cargaban bien, entonces creamos una función específica leerArchivo.
- Memoria dinámica: aparecían fugas porque no liberamos los punteros; lo resolvimos con delete[].
- Orden de operaciones: cuando hacíamos rotación antes de XOR los resultados eran ilegibles, se corrigió invirtiendo el orden en modo desencriptar.
- Diccionario en LZ78: a veces se caía por índices inválidos, lo arreglamos validando antes de usar.

7. Evolución de la solución

- Primera versión: solo lograba leer los archivos y hacer XOR básico.
- Versión intermedia: agregamos RLE y LZ78 pero fallaban los punteros.
- Versión final: ya integramos todo, liberamos memoria correctamente y se validaron los resultados con las pistas.

8. Conclusiones

Este desafío nos sirvió para aplicar de forma práctica:

- Uso de punteros y arreglos dinámicos en C + +.
- Operaciones a nivel de bits (rotación y XOR).
- Implementación manual de algoritmos de compresión.
- Búsqueda de parámetros correctos con prueba y error, usando la pista como validación.

Además, reforzamos la importancia de modular el código y de liberar memoria para evitar errores.