

INFORME DESAFIO I - INFORMÁTICA II

ANÍBAL JOSÉ GUERRA

PRESENTADO POR:

JUAN PABLO MONCADA

ANDRES FELIPE MACA

UNIVERSIDAD DE ANTIOQUIA

FACULTAD DE INGENIERÍA

MEDELLÍN; ANTIOQUIA; 2025

**Introducción:**

Este documento describe la estructura lógica, técnica y funcional del programa desarrollado para resolver el Desafío 1 de la asignatura Informática II, correspondiente al semestre 2025 - 2 del programa de Ingeniería Electrónica de la Universidad de Antioquia.

El programa tiene como objetivo la desencriptación y descompresión de un archivo donde se le aplicará programación inversa para lograr encontrar el texto original , se analizan y revierten transformaciones codificadas en archivos auxiliares .txt, empleando técnicas de encriptado con métodos como el XOR y movimiento a nivel de bits, para la compresión se entregan 2 métodos que son el LZ78 y el RLE, donde toca hallar el método que se usó y descomprimir el texto desencriptado según el método utilizado.

La implementación se realiza en C++ utilizando exclusivamente memoria dinámica, sin uso de estructuras ni STL, respetando las restricciones del reto. A lo largo del documento se detallan las decisiones de diseño, fases del algoritmo, manejo de memoria, funciones utilizadas y un análisis profundo de los riesgos asociados a transformaciones destructivas y ambigüedades lógicas, la información que es entregada puede estar sujeta a cambios en caso de ambigüedades en la creación del programa.

Trabajo realizado por:  
Juan Paulo Moncada Morales - CC 1059699215  
Andres Felipe Maca Narvaez - CC 1061698616

# **Contexto y objetivo:**

Se recibe un mensaje comprimido (por RLE o LZ78) y luego encriptado (rotación de bits a la izquierda n con 0<n<8 y XOR con clave de 1 byte K). También se conoce un fragmento en texto plano del mensaje original. La tarea es:

Identificar: algoritmo de compresión (RLE o LZ78), n y K.

Desencriptar (invirtiendo XOR y rotación).

Descomprimir y reconstruir el texto original.  
Restricciones: C++ en Qt, sin objetos tipo string, sin STL, usar punteros, arreglos y memoriadinámica.

# **Supuestos de formato:**

La “cadena” de entrada del mensaje cifrado se leerá como bytes (p. ej., archivo binario o texto en hex con parser propio).

RLE usa pares (longitud, símbolo) (ambos un byte).

LZ78 usa pares (prefijo, c) con prefijo de 2 bytes (big-endian) + carácter (1 byte), tal como la tabla de ejemplo (byte alto, luego bajo).

El texto original sólo contiene A–Z, a–z y 0–9 (sin tildes, espacios ni “Ñ”).

# **Estrategia de identificación (criba de parámetros)**

Idea central texto conocido.

1. Para cada n en 1…7 y cada K en 0...255:

Desencriptar tentativa: primero XOR con K (porque el cifrado fue ROTL→XOR, así que se revierte XOR→ROTR), luego ROTR n bits *por byte*.

Intentar parsear los bytes resultantes como RLE y, por separado, como LZ78**:**

Descomprimir cada candidato (controlando errores).

Buscar el fragmento conocido dentro del output:

La descompresión no falla (límites válidos, índices LZ78 válidos).

El texto obtenido es alfabético/dígitos solamente.

El fragmento aparece al menos una vez.

Esta búsqueda es tractable: 7 \* 256 = 1792 combinaciones, y cada intento es lineal en el tamaño de los datos.

# **Diseño de memoria y tipos**

Usar char\* para buffers; int para longitudes.

Reusabilidad de buffers para evitar copias; desencriptar en un buffer temporal reutilizable.

**Análisis de patrones de los métodos de compresión LZ78, RNL**

Es posible identificar diferencias claras entre los métodos RLE y LZ78 a partir de los patrones de representación que generan:

* Estructura de RLE:

Produce secuencias en el formato (número + carácter), los números indican cuántas veces se repite un símbolo consecutivo, la salida suele estar compuesta exclusivamente por dígitos seguidos de letras o números ASCII, sin bytes nulos ni estructuras adicionales.

Ejemplo típico: 11W1B12W3B14W.

Validación: cada token debe comenzar con al menos un dígito y terminar en un carácter válido (A–Z, a–z, 0–9).

* Genera pares en el formato:

En este caso, el índice se representa con dos bytes y el carácter con un byte ASCII, formando bloques fijos de tres bytes, al inicio, es común encontrar varias entradas del tipo 0-char, ya que el diccionario comienza vacío y los primeros símbolos se registran con índice 0.

Ejemplo: 0x00 0x00 'A', 0x00 0x00 'B', etc.

Validación: el índice del prefijo nunca debe superar el tamaño actual del diccionario.

* Diferencias clave observables:

RLE: tokens de longitud variable (número con una o más cifras + carácter).

LZ78: bloques constantes de 3 bytes.

RLE: puede comenzar directamente con un número mayor a 1.

LZ78: suele comenzar con secuencias 0-char repetidas.

RLE: sólo caracteres imprimibles (dígitos + letras).

LZ78: incluye valores binarios en los dos bytes del índice.

Si el flujo desencriptado muestra secuencias numéricas seguidas de caracteres es RLE.

Si el flujo presenta bloques de 3 bytes con patrón 00 00 char al inicio es LZ78.

Estadísticamente, encontrar varios tokens 0-char consecutivos al inicio es un fuerte indicador de LZ78.

# **Manejo de errores y validaciones**

En RLELongitud cero, fin de buffer impar, símbolo fuera de rango.

En LZ78 Prefijo 0 permitido; prefijo → fallo; desbordes al copiar/concatenar → fallo.

En cripto rot fuera de 1 ... 7 → ignorar; siempre operar en 8 bits.

// =============================

// Programa principal

// =============================

PROGRAMA

numArchivos ← leerNumeroArchivos()

PARA i = 1..numArchivos

enc ← leerMensaje("Archivo" + i, BIN) // bytes comprimidos y encriptados

pista ← leerPista("Pista" + i) // fragmento conocido

SI vacío(enc) O vacío(pista) →

imprimir("No se pudo leer el archivo " + i)

CONTINUAR

ok ← desencriptarYdescomprimir(enc, pista, i)

SI ok

imprimir("Archivo " + i + " procesado correctamente")

SINO

imprimir("No se pudo desencriptar y descomprimir el archivo " + i)

FIN SI

FIN PARA

generarReporteFinal()

FIN

// =============================

// Helpers

// =============================

desencriptarYdescomprimir(enc, pista, idArchivo) → bool

// intenta encontrar rot y K correctos

// aplica XOR + rotación inversa

// prueba con RLE y LZ78

// si coincide con pista:

// guardarComoTxt(txt, "Original" + idArchivo + ".txt")

// registrarMetodos(idArchivo, algoritmo, rot, K)

// retornar verdadero

// sino retornar falso

leerNumeroArchivos() // entrada usuario

leerMensaje(ruta, formato) // BIN o HEX → bytes

leerPista(ruta) // ASCII → chars

guardarComoTxt(txt, nombreArchivo) // guarda salida

registrarMetodos(id, alg, rot, K) // log de parámetros

generarReporteFinal() // imprime/guarda resumen