# Informe Desafío 1 – Informática II

Universidad de Antioquia – Ingeniería Electrónica  
Curso: Informática II  
Semestre 2025-2  
Estudiante: Joshua Castro Solarte y Nicolas Rubiano  
Profesor: Aníbal José Guerrera Soler

## 1. Contextualización

El problema consiste en recuperar un mensaje original que fue sometido a dos transformaciones consecutivas:  
  
1. Compresión usando uno de dos algoritmos posibles: RLE o LZ78.  
2. Encriptación basada en operaciones a nivel de bits (rotación e XOR).  
  
El reto es identificar qué algoritmo de compresión se usó, cuáles fueron los parámetros de encriptación (rotación y clave XOR), y finalmente reconstruir el mensaje original.

## 2. Análisis del Problema

El problema plantea tres tareas principales:  
  
1. Identificación de parámetros de encriptación:  
 - La rotación puede tener valores entre 1 y 7 bits.  
 - La clave XOR es un número entre 0 y 255.  
 - La búsqueda se realiza probando todas las combinaciones posibles.  
  
2. Reconocimiento del tipo de compresión:  
 - Si el mensaje desencriptado tiene estructura tipo “nLetra” (ejemplo: 12A3B4C), se trata de RLE.  
 - Si el mensaje aparece en forma de pares (índice, carácter), es LZ78.  
  
3. Descompresión:  
 - RLE: expansión de números a repeticiones de caracteres.  
 - LZ78: reconstrucción del texto original usando un diccionario dinámico.

## 3. Diseño de la Solución

El diseño se organizó en módulos independientes:  
  
1. Módulo de encriptación y desencriptación  
 - Funciones para rotar bits a la izquierda/derecha.  
 - Función para aplicar XOR con una clave.  
 - Procedimiento para desencriptar cada byte: primero XOR, luego rotación inversa.  
  
2. Módulo de detección de parámetros  
 - Dos ciclos anidados: uno recorre n = 1…7, otro recorre K = 0…255.  
 - Se genera un texto intermedio y se busca si contiene el fragmento conocido.  
  
3. Módulo de compresión y descompresión  
 - Implementación de RLE: conversión de pares número+carácter en secuencias.  
 - Implementación de LZ78: uso de un diccionario en memoria dinámica para almacenar prefijos.  
  
4. Módulo de verificación  
 - Comprobar que el mensaje reconstruido contiene el fragmento original.  
 - Mostrar parámetros encontrados (n, K, método de compresión).

## 4. Algoritmos Implementados

- Rotación de bits: se mueven los bits hacia un extremo y los que salen entran por el otro.  
- XOR: operación reversible bit a bit; si se aplica dos veces con la misma clave, recupera el valor original.

Byte rotateRight(Byte b, int n);

Byte\* desencriptar(const Byte\* encrypted, int len, int n, Byte k);  
- RLE: leer número, repetir el carácter ese número de veces, concatenar resultado.

int pareceRLE\_ASCII(const Byte\* data, int len);

char\* descomprimirRLE\_ASCII(const Byte\* data, int len, int& outLen);  
- LZ78: a partir de un par (índice, carácter), reconstruir cadenas con un diccionario dinámico.

char\* descomprimirLZ78(const Byte\* compressed, int len, int& outLen);

-Para el manejo de archivos:

Byte\* leerArchivoBinario(const char\* nombre, int& tamano);

char\* leerArchivoTexto(const char\* nombre, int& tamano);

bool contieneFragmento(const char\* texto, int tamTexto, const char\* fragmento, int tamFrag);

-Algorinto de logica principal de resolucion:

bool resolverCaso(const Byte\* encriptado, int tamEnc, const char\* pista, int tamPista,

int& metodo, int& rotacion, int& clave, char\*& textoOriginal, int& tamOriginal);

## 5. Problemas de Desarrollo

Durante la implementación surgieron los siguientes retos:  
  
- Restricción de no usar string ni STL: fue necesario manipular todo con punteros y arreglos de tipo char, lo cual dificulta la gestión de memoria.  
- Manejo del diccionario en LZ78: la implementación requería memoria dinámica para concatenar cadenas, cuidando de no generar fugas de memoria.  
- Búsqueda exhaustiva de parámetros: la cantidad de combinaciones (7 × 256 = 1792) puede hacer lento el programa si no se optimiza, aunque sigue siendo manejable.

## 6. Evolución de la Solución

El desarrollo siguió los siguientes pasos:  
  
1. Etapa inicial: diseño de funciones básicas de rotación y XOR.  
2. Etapa intermedia: creación de los algoritmos de compresión y descompresión RLE y LZ78 de forma independiente.  
3. Etapa avanzada: integración de los módulos en un único flujo que recibe el mensaje encriptado, prueba parámetros, desencripta y descomprime.  
4. Pruebas y verificación: comparación de la salida con el fragmento conocido para validar los resultados.

## 7. Conclusiones

- El reto permitió aplicar los conceptos básicos de C++ relacionados con punteros, arreglos y memoria dinámica.  
- El uso de algoritmos de compresión simples como RLE y LZ78 permitió entender cómo reducir redundancia en textos.  
- La aplicación de operaciones a nivel de bits (rotación y XOR) reforzó el conocimiento de la manipulación binaria.  
- La metodología de prueba y error (fuerza bruta) fue clave para identificar los parámetros de encriptación.