Informe Preliminar Desafio 1

1st Julián Sánchez Ceballos
Facultad de ingeniería
Universidad de Antioquia
Medellín, Colombia
julian.sanchez6@udea.edu.co

2nd Jaider Bedoya Carmona
Facultad de ingeniería
Universidad de Antioquia
Medellín, Colombia
jaider.bedoya@udea.edu.co

***Abstract*—Este documento presenta el análisis y diseño de la solución a un problema de descompresión y desencriptación de un mensaje contenido en un archivo de texto. Dicho mensaje ha sido previamente codificado en dos etapas: primero comprimido mediante LZ78 o RLE, y posteriormente encriptado aplicando una rotación de bits a nivel de byte y una operación XOR con una clave. Se expone la estrategia de ingeniería inversa para recuperar el mensaje original, incluyendo algoritmos, diagramas explicativos y ejemplos prácticos.**

***Index Terms*—Compresión, encriptación, ingeniería inversa, desafío, memoria dinámica, punteros, C++**

INTRODUCCIÓN.

Como parte de la evaluación del curso de Informática 2, se plantea un desafío de ingeniería inversa en C++ orientado al uso de punteros y memoria dinámica. El objetivo es implementar un programa capaz de descifrar archivos comprimidos y encriptados, demostrando habilidades en análisis de problemas, diseño algorítmico y dominio del lenguaje.

El reto consiste en partir de un texto comprimido y encriptado, junto con un fragmento conocido del mensaje original, para determinar los parámetros utilizados en el proceso (algoritmo de compresión, valor de rotación y clave XOR) y, con ellos, reconstruir la totalidad del mensaje original.

Para lograrlo, se definen y analizan los pasos necesarios, priorizando un enfoque modular y un flujo de validación que minimice resultados erróneos.

PROPUESTA DE SOLUCIÓN.

A. *Condiciones y restricciones*

Se conoce que:

- El mensaje fue comprimido utilizando RLE o LZ78.
- Una vez comprimido, se aplicaron dos operaciones de encriptación:
  - Rotación a la izquierda de cada byte en n bits, con $0 < n < 8$.
  - Operación XOR con una clave de un solo byte K.
- Se dispone de un fragmento en texto plano del mensaje original, el cual servirá como referencia para identificar:
  - El método de compresión utilizado.
  - El valor de la rotación n.
  - El valor de la clave XOR K.
  - El mensaje original completo.

B. *Operaciones*

Los métodos de compresión considerados en el desafío son:

1) ***Compresión RLE:*** Es un algoritmo de compresión sin pérdida que funciona mejor en datos con repeticiones consecutivas. Por ejemplo, la secuencia `AAAAABBCCCDDDD` se codifica como `5A2B3C4D`, representando el número de repeticiones seguido del carácter.

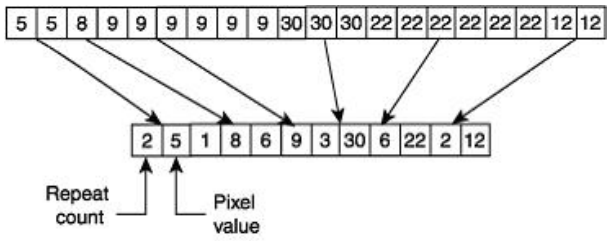

Fig. 1. Ejemplo compresión RLE

2) ***Compresión LZ78:*** A diferencia de RLE, LZ78 construye dinámicamente un diccionario de secuencias vistas. Cada nueva cadena encontrada se codifica como un par `(índice, carácter)`, donde el índice refiere a la secuencia más larga ya presente en el diccionario, y el carácter corresponde al siguiente símbolo en el texto. Esto genera una representación más compacta en casos con patrones recurrentes.

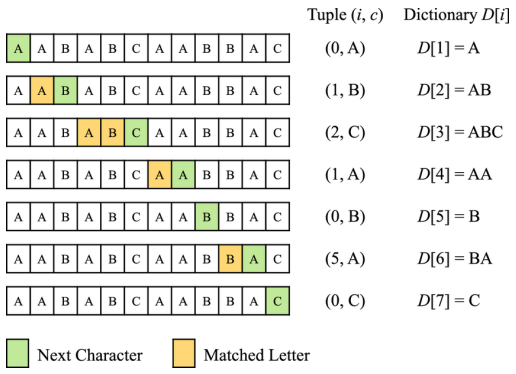

Fig. 2. Ejemplo compresión LZ78

Las operaciones de encriptación aplicadas tras la compresión fueron:

*3) **Rotación de bits**:* Consiste en desplazar los bits de un byte hacia la izquierda o derecha, de forma circular. A diferencia de un corrimiento, los bits que salen por un extremo entran de nuevo por el otro.

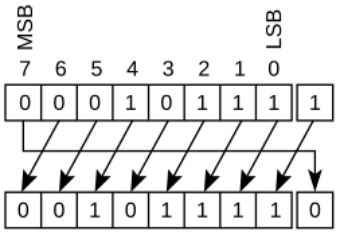

Fig. 3. Ejemplo rotación de un bit

*4) **Operación XOR**:* Es una operación bit a bit con la siguiente regla: $0 \oplus 0 = 0$, $0 \oplus 1 = 1$, $1 \oplus 0 = 1$, $1 \oplus 1 = 0$. Una de sus propiedades es que, aplicando la misma clave dos veces, se recupera el valor original.

```
1011 XOR 0011 = 1000
1000 XOR 0011 = 1011
```

Fig. 4. Ejemplo de una XOR

C. Estrategia mediante fuerza bruta

Conociendo la naturaleza de las transformaciones, se plantea un algoritmo de fuerza bruta que recorra todas las combinaciones posibles de (n, K). Para cada par:

1) Se aplica la operación inversa (XOR y rotación).
2) Se intenta descomprimir con ambos algoritmos (RLE y LZ78).
3) Se valida si el resultado contiene el fragmento conocido.

Este enfoque garantiza encontrar los parámetros correctos al verificar contra un ancla confiable (la pista).

TABLE I
PROCEDIMIENTO DE CIFRADO Y DESCIFRADO

Paso	Cifrado (original)	Descifrado / Verificación
1	Texto plano	Leer fichero encriptado
2	ROL_n sobre cada byte	XOR inverso: $b' = b \oplus K$
3	XOR con clave K	ROR_n sobre cada byte
4	Flujo comprimido (RLE/LZ78)	Comprobar ratio de caracteres imprimibles
5	–	Descompresión LZ78 (si válida)
6	–	Descompresión RLE (si válida)
7	–	Buscar fragmento conocido en el texto
8	–	Si aparece, aceptar (n, K) y método

D. Criterio de validación

Se considera que un resultado es correcto si:

- El flujo desencriptado es interpretable por RLE o LZ78 sin errores de formato.
- El texto descomprimido contiene de manera íntegra el fragmento conocido.

Este criterio evita falsos positivos al no depender únicamente de obtener texto legible.

E. Diagrama de flujo de la solución

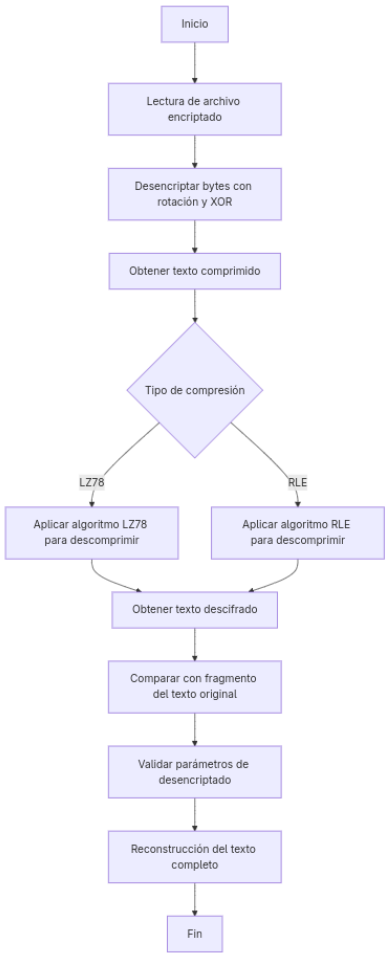

Fig. 5. Diagrama de flujo de la solución

F. Arquitectura de la solución

El sistema se organiza en módulos independientes, empleando memoria dinámica y tipos primitivos en lugar de STL.

*1) **Estructura de directorios**:*

```
/project
  include/
     solver.hpp
     compress.hpp
  src/
     solver.cpp
     compress.cpp
     app.cpp
     make_test_files.cpp
  main.cpp
```

*2) **Descripción de módulos y responsabilidades**:*

- **compress.hpp:** Funciones de manipulación de bits, rotaciones, XOR, compresión y descompresión (RLE y LZ78).
- **solver.hpp:** Encargado de la lectura binaria, desencriptado, descompresión, búsqueda por fuerza bruta y coordinación de operaciones.
- **app.hpp:** Controla el flujo general de la aplicación: entrada de datos, invocación de funciones y escritura de resultados.
- **main.cpp:** Punto de entrada que invoca `app_main()`, manteniendo limpio el ejecutable.

*3) **Lectura de archivos**:* El programa solicita al inicio el número de casos de prueba a procesar (n). Cada caso consta de dos archivos:

- `EncriptadoX.txt`: contiene el mensaje comprimido y encriptado (donde X corresponde al número de caso).
- `pistaX`: archivo con el fragmento de texto conocido para validar el descifrado.

El programa procesa secuencialmente cada pareja de archivos, aplicando la estrategia de fuerza bruta hasta identificar el método y parámetros correctos. Finalmente, reconstruye y muestra el mensaje original.