UNIVERSIDAD DE ANTIOQUIA

1 8 0 3

Facultad de Ingeniería
Departamento de Electrónica y telecomunicaciones

INFORMATICA 2
Preinforme - Desafio 1

Brayan Camilo Silva Porras

Hamilton Alexander Suárez Pérez

2025 - 2

Introducción

El problema consiste en recuperar un mensaje en texto plano que fue sometido a dos procesos consecutivos: compresión (usando RLE o LZ78) y encriptación (rotación de bits a la izquierda por n posiciones en cada byte, seguida de una operación XOR con clave K).

Para recuperar el mensaje original, el proceso debe invertirse cuidadosamente en tres pasos principales:

- **Desencriptación**
  - Aplicar XOR con la misma clave K (ya que XOR es reversible con la misma operación).
  - Rotar los bits a la derecha n posiciones, inverso de la rotación original a la izquierda.
- **Descompresión**
  - Si se usó RLE: expandir secuencias comprimidas como `4A → AAAA`.
  - Si se usó LZ78: reconstruir el diccionario dinámico y recuperar el texto a partir de los pares $(\text{índice}, \text{carácter})$.
- **Identificación**
  - Verificar si el fragmento conocido del mensaje original aparece en el texto recuperado.
  - De ser así, se confirman los parámetros (método, n, K) y se acepta como la solución correcta.

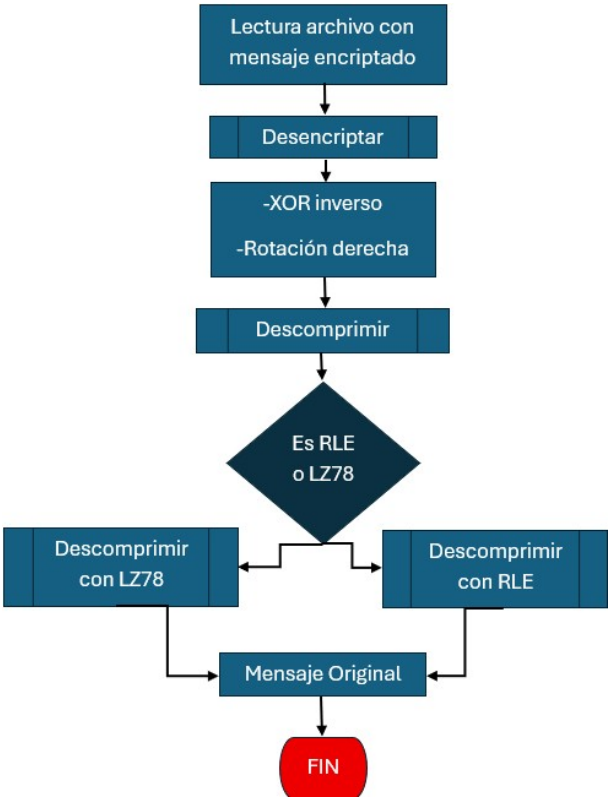
Consideraciones principales

- El espacio de búsqueda es limitado y permite una estrategia exhaustiva: 2 métodos de compresión $\times$ 7 valores posibles de n $\times$ 256 valores de $K = 3,584$ combinaciones.
- No se usan `std::string` ni STL, lo que obliga a diseñar el código con arreglos, punteros y memoria dinámica.
- Los códigos de descompresión que se implementen no serán tan robustos, estarán orientados únicamente a manejar los casos previstos en el enunciado, sin contemplar demasiadas variaciones externas.

Definición de tareas y módulos del proyecto

El proyecto se divide en módulos independientes que colaboran entre sí:

- **Lectura y escritura de archivos (io_utils)**
  - Leer el archivo comprimido y encriptado en un buffer dinámico de bytes.
  - La lectura se hará con `fscanf` para cumplir la restricción de no usar `std::string`.
  - Guardar el mensaje recuperado en un archivo de salida.
- **Funciones de desencriptado (crypto)**
  - `aplicarXor`: aplicar XOR con K a todo el buffer.
  - `rotarNDerecha`: rotar n posiciones a la derecha en cada byte.
- **Funciones de descompresión**
  - `descomprimirRle`: interpretar contadores y expandir secuencias.
  - `descomprimirLz78`: reconstruir cadenas a partir de los pares $(\text{índice}, \text{carácter})$ y un diccionario dinámico.
- **Función de identificación (analyzer)**
  - Probar todas las combinaciones de parámetros (método, n, K).
  - Desencriptar, descomprimir, y buscar el fragmento conocido.
  - Retornar el mensaje original si se encuentra coincidencia.
- **Programa principal (main)**
  - Coordinar todo el flujo: lectura de archivo $\rightarrow$ identificación $\rightarrow$ guardado del mensaje recuperado $\rightarrow$ impresión de parámetros encontrados.



Algoritmos implementados

- **Rotación de bits**
  - `rotarDerecha(byte x, int n)` $= ((x \ll n) \,|\, (x \gg (8-n))) \,\&\, 0xFF$.
  - `rotarIzquierda(byte x, int n)` $= ((x \gg n) \,|\, (x \ll (8-n))) \,\&\, 0xFF$.
  - Usados para revertir la rotación aplicada en la encriptación.
- **XOR**
  - $x = x \oplus K$.
  - Propiedad: $((x \oplus K) \oplus K) = x$.
  - Se aplica sobre todo el buffer como primer paso de desencriptado.
- **RLE (descompresión)**
  - Leer números (en ASCII o binario) como contadores.
  - Repetir el símbolo indicado la cantidad de veces especificada.
  - Ejemplo: `3A2B → AAABB`.
- **LZ78 (descompresión)**
  - Leer tokens: (índice, carácter).
  - Si índice $= 0$: salida $=$ carácter.
  - Si índice > 0: salida $=$ diccionario[índice $- 1$] $+$ carácter.
  - Agregar la nueva cadena al diccionario.
- **Identificación de parámetros**
  - Triple bucle: método $\in \{\text{RLE}, \text{LZ78}\}$, $n \in [1.,7]$, $K \in [0.,255]$.
  - Para cada combinación: desencriptar, descomprimir, buscar fragmento.
  - Si se encuentra, se detiene la búsqueda y se guarda el resultado.

Problemas de desarrollo afrontados

- **Manejo de memoria en LZ78**
  - Restricción: no se puede usar `std::string` ni contenedores STL.
  - **Solución**: usar estructuras con punteros (`byte*`) y longitudes (`size_t`) para almacenar entradas de diccionario.
- **Orden de desencriptado**
  - Duda: determinar si primero aplicar rotación o XOR.
  - **Solución**: se confirmó que el orden correcto es aplicar XOR primero y luego la rotación a la derecha, ya que la encriptación fue rotación a la izquierda seguida de XOR.

- **Control del tamaño de salida**
  - Problema: la descompresión puede generar archivos más grandes que el comprimido.
  - **Solución**: implementar buffers dinámicos que crecen con `realloc` o `new[]` según sea necesario.

Evolución de la solución y consideraciones para la implementación

- **Etapa inicial**
  - Se implementaron funciones básicas de rotación y XOR.
  - Se validó que fueran inversas y produjeran los mismos bytes originales.
- **Etapa intermedia**
  - Se programó la descompresión RLE para casos simples, priorizando funcionalidad antes que robustez.
  - Posteriormente, se implementó la descompresión LZ78 con manejo de diccionario dinámico mediante punteros.
- **Etapa de integración**
  - Se desarrolló la función `identify_and_recover`, que integra desencriptado, descompresión y búsqueda del fragmento conocido en un flujo de prueba exhaustiva.
- **Etapa final prevista**
  - Ajustar la solución en `main.cpp` para coordinar la lectura del archivo, ejecución de la identificación, y guardado del resultado.
  - La implementación final estará enfocada en resolver el desafío planteado, sin incluir validaciones adicionales ni soporte extendido a variantes no mencionadas en el enunciado.