Informe Desafío 1

Tomás Restrepo Saldarriaga, Daniel Velásquez Parra.

29 de septiembre de 2025

1. Introducción

El desafío nos plantea un ejercicio de ingeniería inversa en C++, en el cual se recibe un mensaje comprimido mediante el algoritmo RLE o LZ78. Luego, este es encriptado mediante rotación de bits y la técnica XOR. A partir de un archivo .txt, el cual contiene una pista del mensaje original, debemos identificar el método de compresión, la rotación y la clave XOR, esto con el fin de desencriptar, descomprimir y recuperar el texto completo. El objetivo es implementar algoritmos para dar solución al problema empleando punteros, arreglos y memoria dinámica, no está permitido usar string ni STL.

2. Desarrollo.

2.1. Análisis del problema.

El primer paso consiste en la lectura de los archivos de texto, tanto los mensajes encriptados como las pistas. Dado que se trabaja a nivel de bytes, la lectura se realiza caracter a caracter, almacenando cada uno en un arreglo de memoria dinámica de tipo `char`. El mismo procedimiento se aplica para las pistas asociadas a cada mensaje.

Una vez obtenidos los mensajes temporales, se procede a desencriptar cada uno de ellos mediante un esquema de ciclos anidados. El primer ciclo corresponde a la operación XOR, en el cual se prueban todas las posibles claves en el rango de 0 a 255.

Dentro de este ciclo, se incluye otro que corresponde a la rotación de bits. Aquí, el valor de n varía de 1 a 7, y para cada valor se construye un nuevo arreglo temporal donde se almacena el resultado de rotar cada byte n posiciones. De esta forma, se conserva intacto el resultado del XOR para poder reutilizarlo en las demás rotaciones sin pérdida de información.

Una vez obtenida la versión rotada, se procede a aplicar los dos métodos de descompresión posibles: RLE y LZ78. Cada función de descompresión trabaja sobre el arreglo de entrada, pero no lo modifica, sino que genera un nuevo arreglo dinámico con el mensaje descomprimido. Esto permite reutilizar el mismo arreglo de rotación para ambas descompresiones sin riesgo de alterar los datos.

Finalmente, el mensaje resultante de cada descompresión se compara con la pista correspondiente. Si se encuentra coincidencia, se registran los parámetros de desencriptado (clave XOR, número de rotaciones y tipo de descompresión) junto con el mensaje final.

2.2. Esquema de tareas.

1. **Lectura de archivos de entrada**
   - Solicitar al usuario la ruta del archivo encriptado y del archivo pista.
   - Leer el contenido de cada archivo en arreglos dinámicos (`leerArchivoEncriptado`).
   - Guardar la longitud de cada mensaje y pista en estructuras auxiliares.
2. **Preparación de estructuras de datos**

- Crear los arreglos dinámicos para almacenar los mensajes encriptados y las pistas (`crearArreglo`, `crearArregloLen`).
- Asegurar redimensionamiento dinámico cuando sea necesario (`redimensionarArreglo`).

3. **Proceso de desencriptación**
   - Para cada mensaje:
     a) Probar todas las posibles claves K (0–255).
     b) Para cada clave, probar todas las rotaciones n (1–7).
     c) Aplicar XOR inverso y rotación inversa al mensaje.
     d) Obtener una copia para descompresión.
4. **Descompresión**
   - Intentar con RLE (`descompresionRLE`).
   - Intentar con LZ78 (`descompresionLZ78`).
   - Generar el mensaje descomprimido para cada caso.
5. **Validación con la pista**
   - Comparar si el mensaje descomprimido contiene el fragmento de pista (`CompararMensaje`).
   - Si coincide:
     - Guardar el método de compresión detectado (RLE o LZ78).
     - Guardar los parámetros (clave K, rotación n).
     - Conservar el mensaje original reconstruido.
6. **Presentación de resultados**
   - Mostrar por consola:
     - Método de compresión detectado.
     - Parámetros de encriptación (K y n).
     - Texto desencriptado completo.
7. **Liberación de memoria**
   - Liberar toda la memoria dinámica usada para mensajes y longitudes (`liberarMemoria`).

2.3. Algoritmos implementados.

En el desarrollo de la solución se implementaron varios algoritmos, cada uno con un propósito específico:

- **Lectura de archivos:** se creó una función que abre un archivo binario, lee su contenido y lo almacena en un arreglo dinámico. Además, se guarda la longitud del mensaje para poder procesarlo más adelante.
- **Redimensionamiento de arreglos:** como no se conoce el tamaño exacto de los mensajes al inicio, se implementó un método que aumenta la capacidad del arreglo cuando se llena, copiando los datos existentes en un arreglo más grande.
- **Desencriptación:** se probaron todas las combinaciones posibles de clave K (XOR) y rotación de bits n. Con cada par de valores se genera una versión candidata del mensaje para comprobar si corresponde con la pista.
- **Descompresión RLE:** este algoritmo reconstruye el texto original expandiendo cada carácter la cantidad de veces indicada por el número que lo acompaña.

- **Descompresión LZ78:** se usó un diccionario dinámico que va guardando subcadenas. Cada entrada nueva se forma a partir de un prefijo existente más un carácter adicional, lo que permite reconstruir el mensaje completo.
- **Comparación con la pista:** se diseñó un método que busca el fragmento conocido dentro de cada mensaje descomprimido. Si se encuentra coincidencia, se confirma que se usaron esos parámetros.
- **Liberación de memoria:** al final, todos los arreglos dinámicos son eliminados para asegurar un uso correcto de los recursos de memoria.

2.4. Problemas de desarrollo afrontados.

Durante la implementación del proyecto se presentaron varios retos importantes:

- **Manejo de memoria dinámica:** al trabajar únicamente con punteros y arreglos, fue necesario tener cuidado con la reserva y liberación de memoria. Un error frecuente eran las fugas de memoria o los accesos inválidos, que se resolvieron revisando cada `new` con su respectivo `delete`.
- **Implementación del LZ78:** al inicio fue complicado entender cómo reconstruir el diccionario dinámico en la descompresión. La dificultad principal estuvo en manejar correctamente los índices y garantizar que cada nueva entrada se formara a partir de un prefijo válido más un carácter.
- **Validación de los parámetros de encriptación:** probar todas las combinaciones de K y n generaba resultados incorrectos al comienzo, porque la rotación de bits no estaba implementada en el orden correcto. Se corrigió verificando primero la operación XOR y después la rotación inversa.
- **Comparación con la pista:** en algunos casos el programa no detectaba coincidencias aun cuando existían. Esto se debía a errores en el control de longitudes. Para solucionarlo, se ajustaron los ciclos de comparación para que revisaran todas las posiciones posibles dentro del mensaje descomprimido.
- **Eficiencia del programa:** como se hacen muchas pruebas (256 claves y 7 rotaciones), la ejecución podía volverse lenta. Se resolvió optimizando la creación de copias temporales y liberando memoria en cada iteración.

2.5. Evolución de la solución y consideraciones

La construcción del programa se realizó de manera incremental, dividiendo el problema en etapas claras:

- **Primera etapa:** se implementó la lectura de archivos encriptados y de pistas, verificando que la información quedara almacenada correctamente en memoria dinámica. Esta parte fue la base para poder trabajar con los datos.
- **Segunda etapa:** se diseñó el mecanismo de desencriptación probando claves K y rotaciones n. Inicialmente se implementó solo la operación XOR y posteriormente se incluyó la rotación de bits en el orden correcto.
- **Tercera etapa:** se desarrollaron los algoritmos de descompresión RLE y LZ78. Para RLE el proceso fue más sencillo, mientras que LZ78 requirió más cuidado por la administración del diccionario dinámico.
- **Cuarta etapa:** se integró la comparación con la pista para identificar los parámetros correctos y validar cuál era el método de compresión usado.
- **Última etapa:** se organizó la impresión de resultados y la liberación de memoria dinámica, con el fin de dejar el programa completo y funcional.

Como consideraciones generales, se tuvo en cuenta:

- Mantener el uso de punteros y arreglos en lugar de `string` o estructuras de la STL, cumpliendo los lineamientos del desafío.
- Modularizar el código en funciones para facilitar la depuración y comprensión.
- Asegurar la liberación de toda la memoria dinámica al finalizar el programa.
- Priorizar claridad y corrección sobre eficiencia extrema, dado que la prioridad del desafío era la correcta implementación de los algoritmos.

2.6. Conclusiones

- El proyecto permitió aplicar de manera práctica conceptos fundamentales de C++ como el uso de punteros, memoria dinámica, arreglos y operaciones a nivel de bits.
- Se logró implementar correctamente los algoritmos de descompresión RLE y LZ78, comprobando su funcionamiento mediante la reconstrucción de mensajes originales a partir de archivos encriptados.
- La solución permitió identificar de forma automática los parámetros de encriptación: el valor de la clave K y el número de rotaciones n.
- Se cumplió con los lineamientos del desafío al evitar el uso de `string` y estructuras de la STL, asegurando un manejo manual de la memoria.
- El proceso de desarrollo fortaleció la capacidad de análisis y resolución de problemas, enfrentando retos como el control de longitudes, la validación de pistas y la correcta liberación de memoria.

Referencias

[1] Guerra, A., & Salazar, A. (2025). *Notas de clase – Informática II*. Universidad de Antioquia.