INFORME PLELIMINAR DESAFIO I INFORMATICA II

Manuel Felipe Salazar Burgos

Universidad de Antioquia

Index Terms— Compresión de datos; RLE; LZ78; encriptación; rotación de bits; operación XOR; C++; algoritmos; ingeniería inversa.

1. INTRODUCCIÓN

En este informe se presenta el desarrollo de la solución al *Desafío 1* de la asignatura Informática II. El reto consiste en recuperar un mensaje original que fue sometido a un proceso de **compresión** (usando RLE o LZ78) y luego a un proceso de **encriptación** basado en rotación de bits y la operación XOR. Como única pista adicional se cuenta con un fragmento del mensaje original en texto plano, lo que obliga a aplicar razonamiento lógico, prueba de algoritmos y programación para reconstruir la información completa.

El objetivo principal de la actividad es poner en práctica los conocimientos adquiridos en **C++**, en particular el uso de estructuras de control, punteros, arreglos, memoria dinámica y operaciones a nivel de bits. Además, el ejercicio busca fortalecer la capacidad de análisis y la resolución de problemas en contextos donde no se tiene toda la información disponible desde el inicio.

A lo largo del documento se describen el análisis del problema, los algoritmos implementados, los inconvenientes encontrados y la forma en que la solución evolucionó hasta obtener el resultado final.

2. ANÁLISIS DEL PROBLEMA Y CONSIDERACIONES PARA LA ALTERNATIVA DE SOLUCIÓN PROPUESTA

El primer paso en el desarrollo de la solución fue realizar una revisión de los métodos de **compresión** involucrados en el desafío, en particular RLE y LZ78. Posteriormente, se estudiaron los procesos de **descompresión**, con el fin de comprender cómo recuperar la información original a partir de los datos comprimidos.

De manera similar, se analizó el mecanismo de **encriptación** basado en la rotación de bits y la operación XOR, junto con sus respectivos procesos inversos de **desencriptado**. Para asegurar una comprensión completa, primero se realizaron esquemas y ejemplos a mano en el cuaderno, lo que permitió visualizar claramente el funcionamiento de cada técnica y las relaciones entre ellas.

Una vez entendidos los conceptos, se procedió a implementar cada uno de los algoritmos de manera independiente en códigos y archivos separados. Esta estrategia facilitó el proceso de verificación, ya que permitió validar por etapas la corrección de cada procedimiento antes de integrarlos en una solución completa.

Durante el análisis, surgió una consideración importante: decidir si era mejor **desencriptar todo el mensaje encriptado** y luego descomprimirlo para verificar la pista, o si, en su lugar, resultaba más eficiente **encriptar la pista** y buscarla directamente dentro del texto encriptado. Ambas alternativas fueron exploradas, pero tras varias pruebas se concluyó que la segunda opción era inviable, ya que la búsqueda directa de la pista en el mensaje encriptado no garantizaba resultados consistentes debido a la naturaleza de la transformación por rotación y XOR.

Por lo anterior, la estrategia definitiva consistió en aplicar un proceso completo de **desencriptado** y posterior **descompresión** del texto bajo diferentes combinaciones de parámetros, y únicamente entonces verificar la presencia de la pista conocida dentro del resultado. Este enfoque permitió avanzar de manera ordenada en la identificación del método de compresión, la clave utilizada y el número de bits de rotación, asegurando así la recuperación correcta del mensaje original.

3. ALGORITMOS IMPLEMENTADOS

En esta sección se describen los algoritmos desarrollados en lenguaje C++ para resolver el desafío, aplicando memoria dinámica, punteros y manejo de archivos binarios, de acuerdo con las restricciones planteadas.

3.1. Rotación de bits

La función `rotarDerecha` realiza una rotación de un byte hacia la derecha en n posiciones. A diferencia de un corrimiento lógico, la rotación no pierde información, ya que los bits desplazados por un extremo se reinsertan por el otro. Esta operación es necesaria para revertir la transformación efectuada en la etapa de encriptación.

3.2. Desencriptado de un byte

La función `desencriptarByte` aplica el proceso inverso al de la encriptación original. Primero se ejecuta una operación XOR con la clave K y, posteriormente, una rotación a la derecha en n posiciones. Con esta combinación se recupera el valor original del byte antes de haber sido encriptado.

3.3. Descompresión RLE

El archivo encriptado se procesa en bloques de tres bytes. Tras desencriptar cada bloque, se interpreta de la siguiente forma:

- El segundo byte ($b1$) representa el número de repeticiones (`count`).
- El tercer byte ($b2$) corresponde al símbolo a repetir.

Con esta información, se reconstruye el texto original aplicando el algoritmo de Run-Length Encoding (RLE) inverso, repitiendo el símbolo indicado tantas veces como lo especifica el contador. La salida se almacena de manera incremental en un arreglo dinámico, mientras que un *buffer circular* conserva los últimos caracteres procesados para compararlos con la pista conocida.

3.4. Búsqueda de la pista mediante fuerza bruta

Para identificar los parámetros de encriptación, el programa recorre todas las combinaciones posibles de:

- Rotación n (entre 1 y 7 bits).
- Clave XOR K (valores de 0 a 255).

Para cada par (n, K) se desencripta y descomprime el archivo completo, almacenando los caracteres reconstruidos en memoria dinámica. En paralelo, se utiliza el *buffer circular* para verificar en tiempo real si aparece la pista conocida dentro del texto descomprimido. Cuando se detecta una coincidencia, se guarda el resultado completo en el archivo `salida.txt` y el proceso se detiene.

3.5. Gestión de archivos y memoria

El programa gestiona archivos binarios con `fopen`, `fread`, `fwrite` y libera recursos con `fclose`. Asimismo, emplea memoria dinámica (`malloc`, `realloc`, `free`) y punteros para cumplir con las restricciones del desafío, evitando el uso de objetos `string`, STL o estructuras avanzadas.

4. PROBLEMAS DE DESARROLLO QUE AFRONTÉ

Durante el desarrollo del desafío surgieron varias dificultades que requirieron tiempo y esfuerzo para ser superadas.

En primer lugar, comprender a fondo el funcionamiento de los algoritmos de **compresión y descompresión LZ78** resultó complejo. El uso de diccionarios dinámicos y la reconstrucción de cadenas a partir de prefijos implicaron un análisis detallado y múltiples pruebas antes de lograr una implementación correcta.

Otro reto importante fue la decisión sobre cómo utilizar la **pista conocida**. Se plantearon dos posibles estrategias: encriptar la pista para buscarla directamente en el texto encriptado, o desencriptar el archivo completo y luego comparar con la pista. Explorar ambas opciones tomó tiempo, y finalmente se concluyó que la primera era inviable debido a la naturaleza de las transformaciones, por lo que se adoptó la segunda alternativa.

Finalmente, al aplicar la estrategia de **fuerza bruta** para probar todas las combinaciones de parámetros (n, K), se evidenció que, cuando el texto a procesar es demasiado largo, el tiempo de ejecución aumenta considerablemente. Este comportamiento hace que el método sea poco eficiente en escenarios de gran tamaño, por lo que una posible mejora futura sería incorporar **métodos de búsqueda optimizados**, como los algoritmos de **Knuth–Morris–Pratt (KMP)** o **Boyer–Moore**, que permiten encontrar subcadenas dentro de un texto de manera más rápida y con menor complejidad computacional.

5. CONCLUSIONES

El desarrollo de este desafío permitió integrar y aplicar de manera práctica los conceptos de compresión, encriptación y programación en C++ trabajados a lo largo del curso.

En primer lugar, se logró comprender e implementar los métodos de compresión y descompresión (RLE y LZ78), así como los mecanismos de encriptación y desencriptación basados en rotación de bits y operación XOR. Este proceso reforzó las habilidades de análisis y el uso de operaciones a bajo nivel, punteros, arreglos y memoria dinámica.

Durante el trabajo también se identificaron las limitaciones de enfoques iniciales, como la idea de encriptar la pista y buscarla en el texto encriptado, que resultó inviable. De igual manera, se evidenció que el uso de fuerza bruta permitió obtener resultados únicamente en los casos de compresión **RLE**, mientras que para LZ78 el enfoque resultó impráctico debido a la complejidad de la reconstrucción con diccionario dinámico.

Este hallazgo resalta la necesidad de considerar técnicas más eficientes para búsquedas y validaciones, como los algoritmos de **Knuth–Morris–Pratt (KMP)** o **Boyer–Moore**, que podrían integrarse para mejorar el rendimiento en textos más largos y en escenarios de mayor complejidad.

Finalmente, el proyecto permitió reconocer la importancia de un enfoque iterativo en el desarrollo: iniciar con la comprensión teórica, pasar por la implementación de cada módulo de forma independiente y, posteriormente, integrarlos en una

solución completa. Esta experiencia resultó valiosa para afianzar tanto la lógica de programación como la capacidad de enfrentar problemas con información incompleta.