**INFORME DEL DESAFIO 1**

**JEAN CARLOS PARRA SERRANO**

**EDWIN SETYI DIAZ PILLIMUE**

**UNIVERSIDAD DE ANTIOQUIA**

**INFORMÁTICA 2**

**SEPTIEMBRE DEL 2025**

**MEDELLÍN, ANTIOQUIA**

***Contextualización***

El desafío plantea un problema de ingeniería inversa con archivos que fueron sometidos a dos procesos consecutivos: compresión y encriptación. El mensaje original en texto plano se comprimió usando RLE (Run-Length Encoding) o LZ78 (Lempel–Ziv 78). Luego, el resultado fue encriptado aplicando dos operaciones sobre cada byte:

1. Una rotación de bits hacia la izquierda en una cantidad fija n, con 0 < n < 8.

2. Una operación XOR con una clave K de un byte.

La tarea consiste en revertir este proceso para recuperar el mensaje original, lo cual implica:

- Identificar qué método de compresión se utilizó (RLE o LZ78).

- Determinar los parámetros de encriptación: la rotación de bits n y la clave XOR K.

- Desencriptar el archivo.

- Descomprimirlo para obtener el texto original completo.

Este desafío evalúa el dominio de programación en C++, incluyendo operaciones a nivel de bits, punteros, memoria dinámica y estructuras de control. Para hallar los valores de k y n, el cliente nos entrega un pequeño fragmento que está dentro del archivo encriptado.

***Análisis y diseño***

Para resolver el problema, definimos los pasos principales:

1. Lectura del archivo encriptado y de la pista

- Se abre en modo binario para no perder información.

- Se almacena en un unsigned char\* mediante memoria dinámica.

2. Desencriptación

- Probar todas las combinaciones posibles de n (1 a 7) y K (0 a 255).

- Aplicar la operación inversa: primero XOR con K, luego rotación de bits hacia la derecha n.

3. Identificación del método de compresión

Se aplican filtros sobre la clave k a probar con la rotación de bits n a la derecha, probando al mismo tiempo con la pista, gracias al algoritmo KMP para descartar las posibles combinaciones de una forma rápida y no descomprimir el archivo por cada iteración para buscar la pista.

4. Uso de la pista

- Se cuenta con un fragmento del mensaje original en texto plano, provisto en archivos pistaX.

- Este fragmento sirve como verificación: al descomprimir, el mensaje reconstruido debe contener la pista.

- De esta manera, la pista nos permite validar qué combinación de parámetros (n, K) y qué algoritmo de compresión fueron realmente usados.

- La pista siempre se estará usando en los dos filtros para los métodos, ya que esta está comparando de una forma secuencial con los datos desencriptados y descomprimidos parcialmente.

5. Descompresión

- Con los parámetros correctos ya identificados, se aplica la descompresión:

- Si es RLE, expandir repeticiones.

- Si es LZ78, reconstruir el texto usando el diccionario.

6. Resultado final

- Se guarda en un archivo el mensaje original completo.

- Indicar el método de compresión, la clave K y el valor de n.

***Consideraciones adicionales***

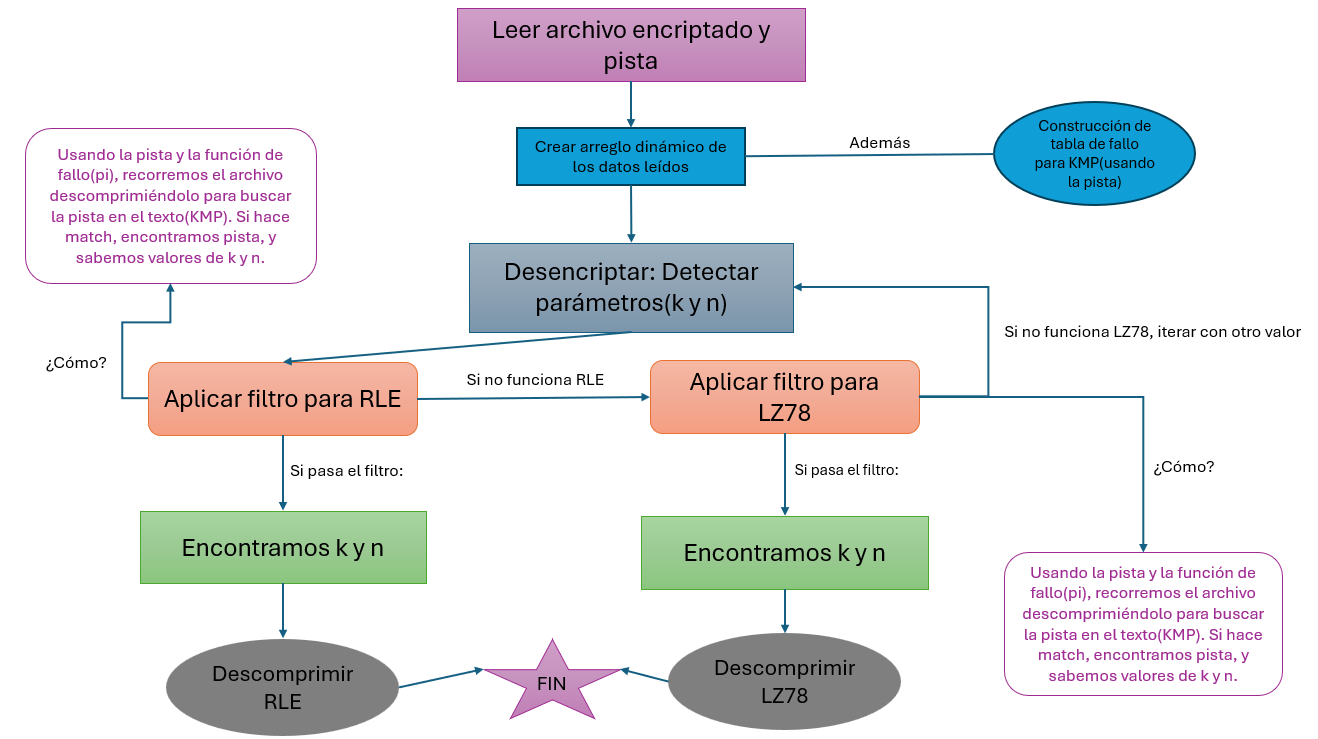
- El texto original se limita a caracteres alfabéticos (A–Z, a–z) y dígitos (0–9), sin tildes, espacios ni símbolos especiales.

- El programa se debe implementar en C++ con Qt, respetando las restricciones:

- No usar std::string, STL ni struct.

- Uso de punteros, arreglos y memoria dinámica.

- El flujo de ejecución pedirá al usuario cuántos casos (n) se van a procesar, y por cada caso abrirá los archivos EncriptadoX.txt y pistaX.



Algoritmos implementados:

|  |  |
| --- | --- |
| Algoritmo | **Definición** |
| Leer archivos | Se crea una función para leer el archivo dinámicamente, usando un puntero que apunta a un arreglo de unsigned char; además se posiciona el apuntador al final para saber el tamaño del archivo y así poder trabajar con estos datos. |
| Crear tabla de fallos | Según la pista, obtenemos la tabla de fallos que es dada por los mayores prefijos, así la tabla de fallos contiene el mismo tamaño que la pista y esta nos servirá para los dos métodos. Esta tabla se usa con la implementación del algoritmo KMP(Knuth-Morris-Pratt). |
| Rotación a la derecha de byte | Se crea una función que rota los bits dada una cantidad n, siendo 0<n<8. Esta función devuelve un unsigned char. |
| Filtro\_RLE | Analiza en tripletas que los dos primeros bytes, sean un número válido y que el otro byte sea una letra. Solo se analiza las tres primeras tripletas para descartar posibles k y n. Además, nunca puede iniciar con 0. |
| RLE\_con\_pista | Se va iterando sobre cada tripleta comparando con la pista, si la pista y el estado del KMP, es decir que la pista coincide con el texto, hacen match, se encontró pista y retorna true. |
| Filtro\_LZ78 | Valida que los índices de prefijo sean coherentes con el tamaño del diccionario. Se hace una comparación similar con el RLE en tema de tripletas, siempre inicia con 0, así que los prefijos no pueden a medida que se lee el archivo, superar el tamaño del diccionario. |
| LZ78\_con\_pista | En general, preprocesa tríos de bytes del arreglo encriptado, desencriptándolos con el número de rotación n y la clave K actuales, seguidamente se emiten estos bytes al algoritmo de búsqueda KMP y se guardan en el diccionario, el proceso se realiza hasta que el algoritmo de búsqueda arroje que se ha encontrado la pista. |
| Descomprimir LZ78 | Desencripta cada byte con la clave k y el número de rotación n que se utilizaron para comprimir el mensaje, reconstruye el índice del diccionario como un entero de 16 bits, usa dos arreglos dinámicos, uno para guardar el índice del prefijo y otro para guardar el último carácter de la entrada, cada nueva entrada se reconstruye recorriendo hacia atrás desde el índice actual hasta el índice 0, finalmente se invierte la frase reconstruida. |
| Descomprimir\_RLE | Se descomprime el archivo siguiendo el orden de los contadores del índice de los dos primeros bytes de cada tripleta y se agrega el carácter al archivo dada las veces que determina el contador para obtener el mensaje desencriptado y descomprimido. |

Algunos problemas que nos enfrentamos:

Desde el inicio tuvimos problemas al leer el archivo y enrutar los archivos en QTCreator. Después sobre los tipos de datos, que devolvían las funciones, se usaron palabras reservadas para no cambiar los contenidos de algunos arreglos. Algunos tipos de datos no eran útiles en casos de desbordamientos, así que se usó por ejemplo *size\_t* para tamaños de archivos. También, la reserva de memoria, qué datos eran necesarios reservar y cuáles no.

En el rendimiento del programa, se había analizado una solución que consistía en la reserva de mucha memoria dinámica y el programa era demasiado lento, así que buscamos nuevas alternativas. Encontramos la solución con los filtros rápidos para los dos métodos y así descartar combinaciones inválidas de inicio. Además el algoritmo KMP nos ayudó a encontrar la pista sin descomprimir todo el archivo con cada clave, sino parcialmente haciendo uso de una gran eficiencia, ya que el primer planteamiento fue del algoritmo de fuerza bruta(naive) que iteraba en una cantidad O(m\*n), siendo n el tamaño de la pista y m el tamaño del archivo. Con el algoritmo KMP, la eficiencia de rendimiento es de O(m+n).

Así pues, los pasos que seguimos fueron:

Se comenzó con funciones básicas de lectura y rotación. Luego se implementó la detección de RLE, más simple. Posteriormente se integró LZ78, que requirió más memoria dinámica. Finalmente se integró KMP y los filtros, consolidando la solución y la posterior descompresión.

***Nota:*** Para la aplicación del desafío se plantearon posibles alternativas. Estas son algunas de ellas.

1. Esta propuesta fue planteada y se desarrolló durante toda la semana 1. Consistía en crear un buffer de trabajo para iterar con él y las posibles claves k y n, e ir eliminando las combinaciones si las claves eran incorrectas. Además, se creaba otro arreglo dinámico, llamado mensaje, donde descomprimía todo el archivo para buscar la pista por fuerza bruta para los dos métodos. Este método era poco eficiente, lento, reservaba mucha memoria que no se usaba o simplemente no servía. En los últimos días de esa semana, se probó el método y tardaba alrededor de 15 segundos en encontrar la pista con solo un archivo, así que no era el camino a seguir.
2. Esta propuesta la planteamos de un orden inverso a la anterior. Consistía en recorrer la pista, comprimirla por los dos métodos, aplicarle los posibles valores de k y n y buscarla en el archivo que nos dieron. Era una idea que íbamos a evaluar pero posiblemente diera muchos falsos positivos si la pista era muy corta o si el archivo era demasiado largo, el proceso sería algo complejo trabajando “a ciegas”.

**Conclusión**

Este proyecto combina conocimientos de algoritmos de compresión, criptografía básica y manejo de bajo nivel en C++. La estrategia general se basa en un proceso de prueba y validación, donde la pista es el elemento clave que permite confirmar los parámetros correctos y el algoritmo de compresión utilizado.