**Informe Final Desafío I**

**Duvan Camilo Aragón Gállego**

**TI. 1033183697**

**David Fernando Revelo Morales**

**C.C 1085909373**

**Informática II**

**Aníbal Jose Guerra Soler**

**Universidad de Antioquia**

**Medellín**

**2025**

**Introducción**

Este proyecto aborda el Desafío I de Informática II, cuyo objetivo es aplicar técnicas de ingeniería inversa para recuperar un mensaje original que ha sido comprimido y encriptado.

A partir de una pista y un archivo encriptado, se busca descubrir los parámetros ocultos de transformación: el tipo de compresión (RLE o LZ78), la clave XOR (k) y el número de rotaciones de bits (n).

El enfoque combina fuerza bruta con filtrado inteligente, validación ASCII y verificación mediante compresión para reconstruir el mensaje original sin conocer el algoritmo fuente.

**Objetivos**

* Descubrir la clave XOR (k) y la rotación de bits (n) aplicadas al texto.
* Identificar el tipo de compresión utilizada en el mensaje.
* Validar la legibilidad del texto desencriptado mediante filtros ASCII.
* Confirmar la presencia de la pista comprimida en el mensaje reconstruido.
* Presentar un sistema modular, eficiente y aplicable para resolver el desafío

**Metodología**

**1. Lectura de Archivos**

El módulo *manejoArchivos.cpp* implementa una lectura binaria, preservando la integridad de los datos y evitando interpretaciones de texto. Devuelve un puntero dinámico *unsigned char\** y su tamaño.

**2. Descubrimiento de Clave y Rotación**

El módulo *encotrarRotacionClave.cpp* implementa la función *RotacionClaveValida* que evalúa sistemáticamente combinaciones de clave XOR y rotación mediante un proceso de filtrado progresivo:

**Estrategia de búsqueda:**

* **Dos fases de exploración:** primero se probó con el byte en la posición 0, ya que comúnmente este valor es 0 y puede revelar la clave de manera inmediata, luego explora todas las claves posibles (0-255).
* **Aplicación secuencial:** Para cada clave candidata, aplica XOR al texto cifrado y luego evalúa rotaciones de 1 a 7 bits a la derecha.
* **Validación de caracteres:** Verifica que los bytes resultantes sean caracteres alfanuméricos (letras mayúsculas, minúsculas o dígitos)

**Proceso de filtrado:**

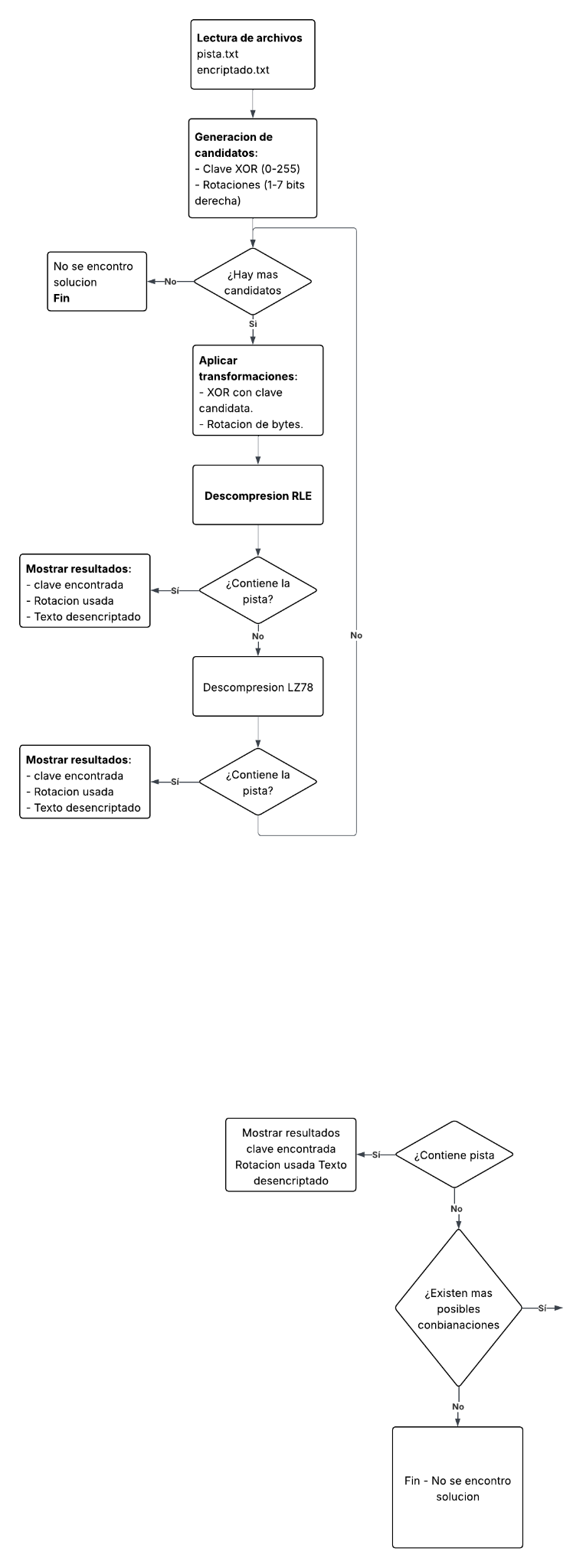
* **Reducción progresiva:** Comienza con 7 rotaciones posibles y descarta aquellas que no producen caracteres válidos en cada iteración.
* **Convergencia rápida:** Reduce el espacio de búsqueda de 1785 combinaciones posibles a pocas docenas de candidatas viables.

**3. Descompresión y Validación**

A las combinaciones candidatas de clave XOR y rotación identificadas en la etapa anterior, se les aplica un proceso secuencial de descompresión y validación:

1. **Descompresión RLE (descompresionRLE):** como primera opción. Si la descompresión RLE produce un texto válido que contiene la pista buscada, el proceso se considera exitoso.
2. **Descompresión LZ78 (descompresionLZ78):** como alternativa cuando RLE no encuentra la pista.
3. **Validación con búsqueda de pista**: La función *buscarSubcadena* verifica la presencia exacta de la pista en el mensaje reconstruido por cada algoritmo.

**4. Diagrama de flujo general**

****

**Desarrollo del Código**

El programa se estructura en seis módulos:

| **Módulos** | **Función Principal** |
| --- | --- |
| *main.cpp* | Orquesta el flujo completo: lectura, desencriptación, descompresión, validación. |
| *manejoArchivos.cpp* | Lectura binaria de archivos en memoria dinámica. |
| *encontrarRotacionClave.cpp* | Descubrimiento de clave y rotación mediante validación progresiva. |
| *compresion.cpp* | Descompresión de RLE y LZ78. |
| *utilidadesCadenas.cpp* | Operaciones básicas de cadenas: longitud, comparación, búsqueda. |
| *utilidadesBytes.cpp* | Manipulación de bytes: separación, unión y rotación. |

**Problemas enfrentados y evolución de la solución**

* **Explosión combinatoria inicial:** probar todas las claves (0–255) y rotaciones (1–7) para cada carácter era ineficiente. Se introdujo la validación ASCII temprana para descartar rápidamente candidatos inválidos.
* **Manejo de memoria dinámica:** al no poder usar string ni STL, hubo que implementar manualmente operaciones de cadena (*longitudCadena, compararCadenas, buscarSubcadena*). Esto aumentó la robustez y el control del programa.
* **Diseño modular:** inicialmente el código estaba centralizado en main, lo que dificultaba depuración. Se dividió en 6 módulos con responsabilidades claras.
* **LZ78 y memoria:** la construcción del diccionario exigía controlar cuidadosamente la memoria asignada y liberada para evitar fugas. Se resolvió con liberación explícita al final de cada ciclo.
* **Validación de resultados ambiguos:** en algunos casos varias rotaciones producían texto legible. Se resolvió comprobando la **pista** como criterio definitivo de corrección.

**Justificación de variables y estructuras de control**

* ***unsigned char\** en lugar de *string***: porque el texto cifrado contiene bytes arbitrarios (0–255) y además estaba prohibido el uso de string/STL.
* **Memoria dinámica (*new[ ], delete[ ]*)**: necesaria para manejar archivos de tamaño variable sin desperdiciar memoria.
* **Ciclos *for* y *while***: empleados para recorrer bytes y validar condiciones hasta encontrar coincidencias.
* **Condicionales *if-else***: usados para filtrar claves inválidas, descartar rotaciones y manejar prefijos en LZ78.
* **Arreglos estáticos *(int rotacionCandidata[7]*)**: se escogieron porque el rango de rotaciones es fijo y pequeño, más eficiente que estructuras dinámicas.

**Resultados**

* **Clave XOR descubierta (k)**: [*valor en hexadecimal*]
* **Rotación de bits (n)**: [*valor entre 1 y 7*]
* **Compresión utilizada**: [*RLE o LZ78*]
* **Texto original reconstruido**: \_\_\_\_\_\_\_\_\_\_\_\_\_ (Recuperado y validado por la pista)

**Conclusiones**

* La estrategia de fuerza bruta guiada por validación ASCII fue efectiva para reducir el espacio de búsqueda.
* La modularidad permite una implementación clara, depurable y escalable.
* El uso de compresión como mecanismo de verificación fue clave para validar la reconstrucción.
* El proyecto integró conocimientos de encriptación, compresión, manipulación binaria y programación estructurada.
* Se logró un sistema robusto, escalable y explicable, cumpliendo con todas las exigencias.