Desafio 1

## Análisis del problema y consideraciones de la solución propuesta

El problema consiste en **reconstruir un mensaje original** que ha sido transformado mediante dos procesos consecutivos:

1. **Compresión**
2. **Encriptación**

Además, contamos con un **fragmento conocido** del mensaje original (una pista que está en el texto9), lo que permite validar si los parámetros elegidos (método de compresión, valor de rotación y clave XOR) son correctos.

Condiciones y observaciones:

* El espacio de búsqueda de parámetros de encriptación es **acotado**: 7 posibles rotaciones × 256 posibles claves = 1792 combinaciones incluso mas porque depende de que método sea ya que si primero probamos con uno y luego con otro serian aun mas combinaciones.
* La validación se realiza comparando la descompresión con el **fragmento conocido**, lo que reduce el riesgo de falsos positivos.
* Restricciones de implementación:
* Se deben usar **punteros, arreglos y memoria dinámica** para todo manejo de datos.

La solución propuesta sigue un enfoque de **fuerza bruta (ir probando los dos métodos) guiada por el fragmento conocido**, pues garantiza encontrar el método correcto y los parámetros. Ya que si en el texto encontramos la pista dada es muy probable que haya sido una ejecución limpia.

## a. Análisis del problema y consideraciones para la alternativa de solución propuesta

El problema planteado consistía en recuperar información de archivos que habían sido **comprimidos** y posteriormente **encriptados** mediante dos pasos consecutivos:

1. **Compresión**:
   * Método **RLE modificado en ternas** → cada secuencia se representa como (basura, conteo, carácter).
   * Método **LZ78 en ternas** → cada secuencia se representa como (índice alto, índice bajo, carácter).
2. **Encriptación**:
   * Aplicación de un **XOR con clave K** sobre cada byte.
   * Posterior **rotación circular a la izquierda de n bits**, con 0 < n < 8.

El reto principal era que, al no conocer de antemano el valor de n, la clave K, ni el método de compresión usado, debíamos diseñar un sistema que:

* Probara automáticamente todas las combinaciones de n y K.
* Reconociera la pista en el texto desencriptado para identificar los parámetros correctos.
* Aplicara la descompresión adecuada (RLE o LZ78) y recuperara el mensaje original.

## b. Esquema de tareas definidas en el desarrollo de los algoritmos

El desarrollo se organizó en las siguientes tareas:

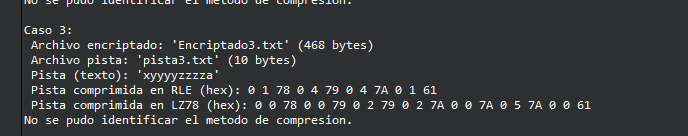
1. **Lectura de archivos**:
   * Implementar funciones para leer archivos binarios y de texto plano.
2. **Módulo de compresión**:
   * Compresor y descompresor RLE en ternas.
   * Compresor y descompresor LZ78 en ternas.
3. **Módulo de encriptación/desencriptación**:
   * Implementar rotación de bits.
   * Implementar operación XOR.
   * Combinar ambas operaciones para desencriptar.
4. **Módulo de búsqueda de claves**:
   * Probar todas las combinaciones de n y K.
   * Comparar la pista comprimida contra los datos desencriptados.
   * Determinar si corresponde a RLE o LZ78.
5. **Integración en el main**:
   * Ejecutar los pasos en orden: lectura → compresión de pista → prueba de claves → descompresión → salida.

## c. Algoritmos implementados

1. **RLE (Run Length Encoding modificado)**:
   * Entrada: texto plano.
   * Salida: ternas (0x00, conteo, carácter).
   * Descompresión: repetir el carácter tantas veces como indique el conteo.
2. **LZ78 (modificado en ternas)**:
   * Entrada: texto plano.
   * Salida: ternas (índice, carácter). El índice se almacena en dos bytes (alto y bajo).
   * Descompresión: reconstruir cadenas usando un diccionario dinámico.
3. **Desencriptación**:
   * Para cada byte: rotarDerecha((byte XOR K), n).
4. **Búsqueda de clave**:
   * Iterar n=1..7 y K=0..255.
   * Desencriptar el archivo y comparar con la pista comprimida.
   * En RLE, ignorar el primer byte basura de cada terna al comparar.
   * En LZ78, comparar directamente las ternas completas.

## d. Problemas de desarrollo que afronté

1. **No encontrar valores de n y K en algunos casos**, el principal problema fue que, en ciertos archivos, el sistema no lograba identificar la combinación correcta de n y K, esto se debió a dos razones:
   * + La pista puede aparecer en cualquier parte del texto, incluso fragmentada o repetida parcialmente.
     + La comparación inicial con memcmp requería coincidencias exactas, lo cual impedía detectar coincidencias parciales.
2. **Complejidad de LZ78** los errores de gestión de memoria ocasionaban fugas o resultados incorrectos en las primeras versiones.
3. **Formato en ternas** inicialmente no estaba claro que en RLE el primer byte de cada terna era “basura”. Esto llevó a fallos en la comparación hasta que se implementó correctamente la lógica de ignorar ese byte.



Validación de formato

* RLE inválido: Al leer contadores muy grandes o secuencias mal formadas, el programa intentaba expandir más memoria de la disponible. Se añadieron comprobaciones de límites.
* Índices inválidos en LZ78: Algunos pares hacían referencia a índices no existentes. Se agregó verificación antes de acceder al diccionario y difieren de los originales.

## e. Evolución de la solución y consideraciones para la implementación

1. La solución comenzó con un prototipo que solo manejaba RLE. Luego se extendió para soportar LZ78 aunque no esta tan definido y puede que cambie caracteres asi como puede no encontrar los valores de n y k.
2. La búsqueda de n y K evolucionó de una comparación estricta (memcmp) a una comparación adaptada al formato de cada compresor.
3. todas las estructuras dinámicas (diccionarios, buffers de salida) se implementaron con new[] y delete[] para que toda la memoria dinámica que se haya implementado pueda eliminarse sin problemas.
4. La modularidad fue clave: separar cada módulo (lectura, compresión, des encriptación, búsqueda, descompresión) permitió depurar errores más fácilmente y también hizo que todo fuera más ordenado.
5. Como consideración futura, se recomienda:

Terminar de ver lz78, asegurarse que se encuentren los valores de k y n