## **1. Contextualización del problema**

En este desafío se plantea un escenario de **ingeniería inversa**: se nos entrega un mensaje que ha sido sometido a dos procesos consecutivos:

1. **Compresión** con uno de dos algoritmos:  
   * **RLE (Run-Length Encoding):** útil en datos con muchas repeticiones consecutivas.
   * **LZ78:** usa un diccionario dinámico que almacena subcadenas ya vistas y codifica nuevas entradas como (índice, carácter).
2. **Encriptación** aplicando:  
   * Una rotación de bits a la izquierda (valor n, con 0 < n < 8).
   * Una operación XOR con una clave de un byte (K).

El resultado disponible es un archivo binario encriptado. Como única pista adicional se conoce un **fragmento en texto plano** del mensaje original.

El reto consiste en, a partir del archivo encriptado y de la pista, **reconstruir el mensaje original completo**, e identificar:

* El método de compresión usado (RLE o LZ78).
* Los parámetros de encriptación (n y K).

Este problema simula situaciones reales donde se recibe información transformada sin conocer los parámetros exactos de codificación, y se debe deducir la configuración correcta.

## **2. Análisis de la posible solución**

Para resolverlo hay que invertir el proceso aplicado al mensaje:

1. **Desencriptar:**
   * Se sabe que la encriptación es reversible.
   * El orden inverso es: primero XOR con K, luego rotar a la derecha n bits.
   * Como no se conocen n ni K, la estrategia es **probar exhaustivamente** todos los valores posibles:  
     + n ∈ [1..7]
     + K ∈ [0..255]
2. **Descomprimir:**
   * Con cada desencriptado, intentar **RLE** y **LZ78**.
   * Si la descompresión produce un texto que contiene la pista conocida, se concluye que los parámetros son correctos.
3. **Validación mediante pista:**
   * La pista funciona como “firma” que permite verificar cuál combinación de n, K y algoritmo es la correcta.

### **Consideraciones técnicas**

* Restricciones de implementación:  
  + Sin string ni STL.
  + Uso obligatorio de punteros, arreglos y memoria dinámica.
  + Los archivos se leen y escriben con fstream.
* Eficiencia: aunque la búsqueda de parámetros es exhaustiva (7 × 256 = 1792 combinaciones), es manejable dado el tamaño de los archivos.
* Robustez: se debe tolerar que un intento de descompresión falle, descartando la combinación y continuando.

## **3. Diseño de la solución**

### **Entradas y salidas**

* **Entradas:**
  + EncriptadoX.txt → mensaje comprimido y encriptado.
  + pistaX.txt → fragmento del mensaje original.
* **Salida:**
  + SalidaX.txt → reporte con:  
    - Método de compresión usado.
    - Parámetros de encriptación (n,K).
    - Mensaje original completo.

### **Arquitectura por módulos**

1. **Módulo de lectura/escritura de archivos**
   * Funciones para cargar datos binarios y pistas de texto desde ficheros.
   * Escritura de resultados en SalidaX.txt.
2. **Módulo de encriptación/desencriptación**
   * Función de rotación de bits.
   * Función de desencriptar un buffer dado n y K.
3. **Módulo de compresión**
   * Implementación de descompresión **RLE** en formato <num><caracter>.
   * Implementación de descompresión **LZ78** en formato (2 bytes de prefijo, 1 byte carácter).
4. **Módulo de validación**
   * Verifica si la salida descomprimida contiene la pista.
5. **Módulo de control principal**
   * Itera sobre n y K.
   * Para cada combinación, prueba RLE y LZ78.
   * Si encuentra coincidencia con la pista, detiene la búsqueda, guarda el resultado e informa al usuario.

### **Flujo de ejecución**

1. Preguntar al usuario cuántos casos (n) se van a procesar.
2. Para cada caso:  
   * Leer “EncriptadoX.txt” y “pistaX.txt”.
   * Ejecutar búsqueda exhaustiva de parámetros.
   * Cuando se encuentra la combinación válida, escribir “SalidaX”.