Desafio 1

Juan pablo rivero garay

Posible solucionamiento del desafio por funciones que debe cumplir el código.

Paso a paso

**Compresión** (uno de los dos métodos):

**RLE (Run-Length Encoding)**: reemplaza secuencias repetidas por (cantidad + símbolo).

**LZ78**: usa un diccionario dinámico y pares (índice, caracter).

**Encriptación**:

**Rotación de bits** (izquierda, n posiciones, con 0 < n < 8).

**XOR con clave K (1 byte)**.(Ambos son reversibles.)

Posibles enfoques para resolver el desafío

1

Probar todas las combinaciones posibles de parámetros de encriptación (n y K) sobre el mensaje comprimido y verificar si, tras desencriptar y descomprimir, aparece el fragmento conocido.

Pasos:

* 1. Iterar valores de n en [1..7].
  2. Iterar valores de K en [0..255].
  3. Desencriptar el mensaje con cada combinación.
  4. Intentar descomprimir con RLE → verificar fragmento.
  5. Intentar descomprimir con LZ78 → verificar fragmento.

Hay que tener cuidado con la memoria

Conclusión

Vamos ha probar el primer método y en caso función tratar de mejorar la eficiencia y uso de memoria para evitar fugas

## 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 esta 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.
* La validación se realiza comparando la descompresión con el **fragmento conocido**, lo que reduce el riesgo de falsos positivos.
* Restricciones de implementación:
* 1) **Sin uso de string, STL ni estructuras**.
* 2) 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.

1. Esquema de tareas definidas en el desarrollo de los algoritmos
2. **Entrada/salida de datos**
   * Leer número de casos a evaluar.
   * Leer archivos EncriptadoX.txt y pistaX en memoria dinámica.
3. **Desencriptación**
   * Probar todas las combinaciones (n, K).
   * Aplicar XOR y rotación inversa sobre cada byte del mensaje encriptado.
4. **Descompresión**
   * Intentar interpretar el buffer desencriptado como **RLE**.
   * Si falla, intentar como **LZ78** con un diccionario dinámico.
5. **Verificación**
   * Buscar el **fragmento conocido** dentro del resultado descomprimido.
   * Si aparece, confirmar parámetros (n, K, método).
6. **Salida de resultados**
   * Reportar método usado, rotación y clave.
   * Mostrar o guardar mensaje original reconstruido.(todavía no sabemos si guardar en un nuevo archivo o imprimir en pantaña por la consola)

d. Problemas de desarrollo afrontados

Manejo de memoria dinámica

* Fugas de memoria: Ocurrieron al olvidar liberar buffers auxiliares (dec\_buf, out\_buf, diccionario de LZ78). La solución fue implementar una rutina de liberación al final de cada caso de prueba.
* Crecimiento del buffer en descompresión: Al expandir símbolos repetidos en RLE, el tamaño del mensaje crece. Se solucionó con realloc dinámico y verificaciones de error.
* Fragmentación del diccionario en LZ78: En la primera versión, cada entrada del diccionario se almacenaba con malloc individual, lo que generaba fragmentación y pérdida de eficiencia. Se reemplazó por un esquema de buffer contiguo (pool de memoria) con offsets y longitudes.

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.

e. Evolución de la solución y consideraciones

* Implementar logs de depuración para seguir el crecimiento de los buffers y uso de memoria.
* Optimizar la búsqueda de parámetros usando el fragmento conocido directamente en la fase de desencriptación parcial.
* Incorporar un sistema de pruebas automáticas con casos simples de RLE y LZ78 para verificar integridad del sistema antes de procesar casos grandes.

**Conclusión**

La solución evoluciona desde una versión básica con problemas de memoria hacia una implementación robusta y modular.