Plan de Trabajo – Ingeniería Inversa de un Mensaje Comprimido y Encriptado

Diego Andrés Páez Mendoza, Jesús Alberto Córdoba Delgado.

1. Análisis del Problema y Consideraciones

1.1 Contexto y objetivos

Se debe reconstruir un mensaje de texto original a partir de una versión que fue primero comprimida y luego encriptada. El objetivo es demostrar dominio de C++ (manipulación de bits, memoria dinámica, punteros y arreglos) implementando los métodos de compresión y las operaciones criptográficas, y recuperando el texto original.

Cadena de transformaciones:

- **Compresión:** RLE o LZ78 (ambos reversibles).
- **Encriptación:** rotación a la izquierda de n bits (1–7) por byte, seguida de XOR con clave K (1 byte).

Disponibles: (i) el mensaje final comprimido+encriptado y (ii) un fragmento conocido del mensaje en claro. Objetivos: identificar el método (RLE/LZ78) y los parámetros (n, K), desencriptar, descomprimir y obtener el texto completo.

1.2 Restricciones y requerimientos no funcionales

- Implementación en C++ (Qt consola).
- Prohibido usar STL (incluido std::string) y contenedores estándar.
- Usar únicamente arreglos, punteros y memoria dinámica.
- Elegir tipos adecuados para eficiencia y justificar decisiones (p. ej., uint8_t, uint16_t, size_t).

Implicaciones: se requiere una mini-biblioteca de utilidades para cadenas estilo C (longitud, copia, concatenación, búsqueda), y una estructura de datos personalizada para el diccionario LZ78 (sin std::map/std::vector).

2. Esquema de Tareas y Arquitectura de la Solución

2.1 Visión general de módulos

- Módulo de Carga: lectura del mensaje cifrado y del fragmento conocido.
- Módulo de Identificación: búsqueda exhaustiva de (n, K) y del método (RLE/LZ78).
- Módulo de Desencriptación: aplica XOR(K) y luego rotación derecha n a cada byte.
- Módulo de Descompresión: RLE.decode o LZ78.decode para recuperar el texto.
- Módulo de Salida: imprime texto original y parámetros hallados.

Secuencia de operaciones (transformación e inversión):

Proceso	Operación 1	Operación 2
Encriptación	Rotación izquierda n bits	XOR con clave K
Desencriptación	XOR con clave K	Rotación derecha n bits

2.2 Lógica de identificación (búsqueda controlada)

El espacio de búsqueda es acotado (n=1 ... 7, K=0 ... 255). Para cada combinación, se desencripta y se intenta descomprimir con RLE y LZ78. Si la descompresión es válida y el resultado contiene el fragmento conocido, se consideran hallados método y parámetros; luego se reconstruye el mensaje completo.

1. Iterar K=0 ... 255 y, para cada K, n=1 ... 7.
2. Desencriptar el buffer con XOR(K) y rotación derecha n.
3. Intentar RLE.decode (validar formato). Si falla, intentar LZ78.decode (longitud múltiplo de 3 y prefijos válidos).
4. Si el texto obtenido contiene el fragmento conocido, aceptar (método, n, K) y finalizar.

Alternativa equivalente: en lugar de simular con el fragmento, validar decodificación completa y buscar el fragmento en el resultado.

3. Diseño preliminar (sin STL)

3.1 Operaciones de bits/bytes

Rotación circular de 8 bits (derecha): resultado = (b >> n) | (b << (8 - n)); usar uint8_t.

XOR por byte: reversible, aplicar primero XOR(K) y luego rotación derecha n al descifrar.

3.2 Estructuras y utilidades

Cadenas estilo C (char*): implementar strlen, strcpy, strcat, strstr/contains simples. Para LZ78, usar una estructura ligera (por ejemplo, entradas (parent, c) con índices uint16_t) para reconstruir frases escalando por padres.

4. Consideraciones y riesgos

Gestión de memoria dinámica: prevenir fugas; política clara de ownership; realloc en crecimientos por factor 2.

Casos límite LZ78: validar prefijos, manejar profundidad de prefijos, y longitud de salida acumulada.

5. Evolución y plan de pruebas

5.1 Fases y commits

- Fase 1 – Algoritmos base: rotación/XOR, RLE.decode, LZ78.decode.
- Fase 2 – Identificación y utilidades: búsqueda (n,K), helpers de cadenas.
- Fase 3 – Integración y pruebas end-to-end; mejoras de memoria y reporte final.

5.2 Pruebas unitarias y de integración

- Rotación/XOR: comprobar reversibilidad.
- RLE.decode: casos con runs largos y entradas inválidas.
- LZ78.decode: validar con ejemplos canónicos (pares (índice, c)).
- Identificación: vectores sintéticos que verifiquen detección del método y parámetros.

6. Diagrama de flujo