Pre-informe – Desafío 1

Informática II – 2025-2

Fecha: 19 de septiembre de 2025

1. Análisis del problema

El desafío consiste en reconstruir un mensaje original que primero fue comprimido (por RLE o LZ78) y luego encriptado mediante rotación de bits a la izquierda (0 < n < 8) seguida de una operación XOR con una clave de un byte. Solo se dispone de: el mensaje comprimido + encriptado y un fragmento conocido del mensaje original en texto plano. El objetivo es identificar automáticamente: 1) el método de compresión utilizado, 2) los parámetros de encriptación: cantidad de bits de rotación (n) y la clave XOR (K), y 3) desencriptar y descomprimir para obtener el texto original completo.

2. Estrategia de solución propuesta

Se empleará un enfoque de fuerza bruta para probar todas las combinaciones posibles de parámetros hasta encontrar aquella que contenga el fragmento de texto plano conocido.

2.1 Fase de desencriptación

1. Prueba de XOR directo: recorro las 256 claves posibles aplicando XOR. Si el resultado preliminar contiene el fragmento, lo marco como candidato. 2. Si no hay coincidencia, pruebo rotación: para cada rotación de 1 a 7 bits a la izquierda aplico la rotación inversa (derecha) y combino con la búsqueda de la clave XOR. 3. Combinación (mix): pruebo las dos operaciones en el orden inverso cuando sea necesario.

2.2 Fase de descompresión

Una vez desencriptado el mensaje: 1) Intento descompresión RLE. 2) Si no hallo el fragmento, intento descompresión LZ78. La coincidencia del fragmento determina el método correcto de compresión.

3. Arquitectura y diseño

Para evitar una larga cadena de if, usaré un sistema de selección con switch-case, que facilitará: elegir el método de descompresión (RLE o LZ78) y seleccionar la combinación de operaciones de desencriptación a probar. Módulos previstos: Desencriptación, Descompresión, Verificación y Controlador principal.

4. Algoritmos a implementar

Rotación de bits: corrimiento circular de 1 a 7 posiciones. XOR: operación bit a bit con clave de 8 bits. RLE: compresión y descompresión por conteo de repeticiones. LZ78: compresión y descompresión con diccionario dinámico.

5. Problemas previstos y consideraciones

Eficiencia: la búsqueda por fuerza bruta puede requerir hasta 256 claves XOR × 7 rotaciones; optimizaré el orden de prueba para reducir el tiempo. Restricciones de C++: no se podrán usar std::string ni STL; usaré arreglos de char, punteros y memoria dinámica. Control de memoria: debo gestionar cuidadosamente la memoria dinámica para evitar fugas.

6. Evolución esperada de la solución

1. Implementar y probar cada módulo por separado. 2. Integrar en el sistema de fuerza bruta. 3. Validar con el fragmento conocido.

7. Preguntas y respuestas surgidas tras la lectura del documento

1) ¿Debo primero desencriptar y después descomprimir? Sí. El mensaje fue primero comprimido y luego encriptado; por lo tanto, la inversión de pasos exige desencriptar primero y descomprimir después. 2) ¿Cómo puedo saber cuánta memoria dinámica estoy usando? Puedes monitorear con herramientas de depuración (Valgrind, AddressSanitizer) o mantener contadores de bytes reservados/liberados en tu propio código. 3) ¿Cómo puedo crear el diccionario para descompresión en LZ78? Usa un arreglo dinámico de cadenas (o punteros a bloques de char) que vayas llenando a medida que lees cada par (índice, carácter). 4) ¿Cómo puedo saber cuál fue el método de encriptación, rotación de bits o XOR? El reto usa ambos: primero rotación y después XOR. La tarea es encontrar la cantidad de bits y la clave XOR correctos. 5) ¿Cómo puedo saber el valor correcto de n en la rotación? Prueba los valores de 1 a 7 y verifica si tras descomprimir aparece el fragmento conocido. 6) ¿Qué pasa si me dan una pista incompleta, como 5A cuando el mensaje real es 10A? Debes buscar coincidencias parciales; asegúrate de que la comparación permita encontrar la pista aunque se repita más veces. 7) ¿Qué hago si obtengo un resultado parcial falso (otro 5A por error)? Continúa con la descompresión completa y verifica que el resto del texto concuerde; un falso positivo se descarta si el resto no coincide. 8) ¿Cómo puedo recorrer el arreglo para comprobar que no haya caracteres inválidos? Recorre byte a byte y valida que cada valor esté en los rangos permitidos (A–Z, a–z, 0–9). 9) ¿Cuáles son las limitaciones de la rotación y del XOR como métodos de encriptación? Son fácilmente vulnerables a fuerza bruta; su seguridad depende solo de la clave y el número de bits, que tienen un espacio de búsqueda pequeño. 10) ¿Qué pasa si la pista está al final del mensaje? Asegúrate de comparar el fragmento con toda la salida descomprimida, no solo con una parte inicial. 11) ¿Puedo comprimir la pista para compararla con cadenas comprimidas? Sí, pero es más seguro descomprimir y comparar en texto plano para evitar ambigüedades de compresión. 12) ¿Qué otros métodos de compresión debo usar? Solo RLE y LZ78, que son los exigidos en el desafío. 13) ¿Por qué en LZ78 aparece el 0 y en RLE no? En LZ78, el índice 0 representa un prefijo vacío; en RLE no se usan índices, solo conteos. 14) ¿Qué hago si el profesor me da un mismo código (3A4G8T…) y me dice que puede ser RLE o LZ78? Intenta descomprimir con ambos y verifica cuál reconstruye un texto coherente. 15) ¿El profesor me dará un texto plano o siempre comprimido? El archivo de entrada siempre estará comprimido y encriptado. 16) ¿Cómo se comprime en LZ78? Se crean pares (índice, carácter) añadiendo cada nueva subcadena al diccionario dinámico. 17) ¿Por qué la cadena no me da directamente el valor que acompaña a la letra en LZ78? Porque el valor de índice representa la posición en el diccionario, no un número que aparezca literalmente en la cadena. 18) ¿Cómo va aumentando el número que antecede a la letra en LZ78? El índice se incrementa a medida que se agregan nuevas entradas al diccionario. 19) Si aparece una C, ¿queda (0,C) y luego CB se guarda como (1,B)? Sí. El primer par crea la entrada “C”; el segundo indica que la nueva subcadena es “CB” que parte de la entrada 1. 20) ¿Debo usar 2 bytes para el número en LZ78? Depende del tamaño máximo esperado; puedes usar 2 bytes (16 bits) si prevés más de 255 entradas. 21) ¿Puedo usar los 2 bytes de forma estática? Sí, siempre que definas un tipo de dato fijo como unsigned short. 22) ¿Cómo puedo hacer que el código decida automáticamente cuánta memoria reservar? Implementa una

estrategia de crecimiento: reserva un bloque inicial y duplica su tamaño cuando se llene. 23) ¿Puedo hacer un switch automático que pruebe todos los métodos de encriptación y descompresión en cascada? Sí, puedes programar un bucle que recorra todos los casos y use switch-case para llamar cada método. 24) ¿Cómo puedo minimizar el uso de memoria en todo el proceso? Libera buffers intermedios en cuanto no sean necesarios y reutiliza arreglos cuando sea posible. 25) ¿Puedo usar el código ASCII como diccionario interno de la máquina? Sí, los primeros 256 valores de ASCII pueden ser tu base de caracteres iniciales. 26) ¿Qué hago si aparecen comas y paréntesis (validar que sean cíclicos)? Verifica que los caracteres estén dentro de los rangos permitidos; si no, descarta el resultado como inválido.