Informe del Desafío 1

Edwin Piedrahita Piedrahita

Facultad de ingeniería, Universidad de Antioquia, Informática 2

Prof. Augusto Enrique Salazar Jiménez

28 de septiembre de 2025

- **Análisis del desafío**

Como primero se tienen algunas entradas conocidas, como un mensaje comprimido y encriptado junto con un fragmento del texto original, ya por la parte que no se conoce está el método de compresión que puede ser por **RLE** o por **LZ78,** también se desconocen los parámetros de encriptación **(n)** que es el valor de la **rotación de bits**, y **(k)** que es el valor de la clave usada para la operación **XOR.**

Como segundo, se puede notar que para hallar los valores y las formas distintas en las que se puede recuperar el texto original son demasiadas, ya que hay dos formas distintas en las que se pudo usar la compresión, siete formas en las que se puede aplicar la rotación de bits, y 256 maneras de aplicar la clave correcta para realizar el XOR, lo cual nos daría una cantidad de 2 x 7 x 256 = 3584 combinaciones posibles para recuperar el texto original, lo cual dificulta encontrar y hallar los diferentes valores y combinaciones usadas para la compresión y encriptación.

También se tienen algunos desafíos como lo son las restricciones del lenguaje, como el no usar la STL, estructuras ni string, además de que se debe usar obligatoriamente la memoria dinámica, por otra parte, la complejidad de los algoritmos y operaciones a nivel de bits.

Base a lo dicho anteriormente lo que se tiene como objetivo es recuperar el mensaje original aplicando las operaciones en el orden correcto y con los valores correctos, en otras palabras, lo que se debe hacer es aplicar ingeniería inversa a los procesos realizados anteriormente para la compresión y encriptación del texto original.

- **Diseño de la solución**

La idea sería primero implementar una función que detecte cuales son los parámetros, como quien es **(k),** quien es **(n),** y cuál es el método que se uso para la descompresión, lo cual ayudara bastante para saber que combinaciones se usaron para la compresión y encriptación del texto original, para lo cual se usara él fragmento de texto original y así poder saber cuál combinación fue la que dio éxito.

Como segundo sería agregar una función que permita desencriptar el mensaje, en la cual se usara la operación **XOR** con su respectiva clave **(k),** además de la rotación de bits a la derecha una cantidad de **(n)** posiciones, ya que para la encriptación se uso la rotación de bits a la izquierda, las cuales se pueden realizar en funciones separadas.

Como tercero se realizará una función de descompresión, la cual va permitir obtener el mensaje original, acá también se pueden crear dos funciones, una para la descompresión

por **RLE** y otra por **LZ78,** para que la función de descompresión sea más simple y fácil de entender**.**

Por último, se implementará una función, la cual manejará todo esto de forma más directa, la cual nos dará el mensaje encriptado, y el fragmento de texto original, generando mayor optimización para realizar estas operaciones, y así disminuir en gran cantidad el uso de la memoria.

- **Algoritmos implementados:**

  Rotación de bits a la derecha (rotarDerecha):

  Recibe un byte y un número de posiciones.

  Aplica un desplazamiento circular: los bits que salen por la derecha reentran por la izquierda.

  Permite revertir la rotación izquierda usada en la encriptación.

  Desencriptación (desencriptar):

  Recorre byte a byte el mensaje encriptado.

  Primero aplica la operación XOR con la clave candidata.

  Después rota los bits a la derecha con el valor de rotación candidato.

  Devuelve un nuevo arreglo dinámico con el mensaje desencriptado.

  Descompresión por RLE (descomprimirRLE):

  Lee números que representan la cantidad de repeticiones y el símbolo asociado.

  Reconstruye el mensaje expandiendo cada par (longitud, símbolo).

Usa punteros y memoria dinámica para manejar la salida.

Descompresión por LZ78 (descomprimirLZ78):

Implementa un diccionario dinámico (arreglo de punteros) donde cada entrada corresponde a una subcadena.

Cada trío de entrada contiene: (índice del prefijo, carácter nuevo).

Se reconstruye el mensaje concatenando el prefijo del diccionario con el nuevo carácter y se inserta en la salida.

El diccionario se maneja con memoria dinámica, liberando cada entrada al finalizar.

Verificación de fragmento (contieneFragmento):

Busca dentro de un texto si existe el fragmento de pista.

Recorre carácter por carácter comparando subcadenas.

Lectura de archivos (leerArchivo):

Abre un archivo en modo binario.

Carga su contenido en memoria dinámica (arreglo de caracteres).

Retorna el contenido junto con su longitud.

Búsqueda de parámetros y método (encontrarParametrosYMetodo):

Prueba todas las combinaciones posibles de parámetros:

Rotaciones entre 1 y 7.

Claves XOR entre 0 y 255.

Ambos métodos de descompresión (RLE y LZ78).

En cada intento, aplica desencriptación + descompresión.

Si el texto resultante contiene el fragmento conocido, se guardan los parámetros correctos y el método.

Función principal (main):

Lee el mensaje encriptado y la pista desde archivo.

Llama a la búsqueda de parámetros.

Muestra en pantalla el método, la rotación y la clave encontrados.

Reconstruye y muestra el mensaje original completo.

Libera toda la memoria dinámica usada.

- **Problemas de desarrollo:**

  Restricciones del lenguaje:

  No se permitió usar string ni STL, lo que dificulto la manipulación de cadenas, lo cual se pudo resolver trabajando directamente con arreglos dinámicos de char.

  Depuración:

  El debugging fue más complejo por las restricciones del lenguaje, por lo que se requirió imprimir resultados intermedios para validar el correcto funcionamiento.

  Descompresión LZ78:

  Fue uno de las partes más complejas debido a la gestión del diccionario, la necesidad de codificar correctamente los prefijos y el riesgo de errores por parte de la memoria.

  Lectura de archivos:

Fue complicado debido a que los archivos dados contenían algo de corrupción, ya que aparentemente parecían compresiones mezcladas entre RLE y LZ78, debido al formato en el que salían al desencriptarse.

Manejo de memoria dinámica:

El uso de punteros me llevo a implementar manualmente la gestión de memoria (reservarla y liberarla), principalmente en la descompresión LZ78 donde el diccionario crece dinámicamente.

- **Evolución de la solución**

  Parte inicial:

  Se implementó primero la función de rotación y la operación XOR para tener la base del desencriptado.

  Desencriptación completa:

  Se integraron ambas operaciones en la función desencriptar, asegurando que el proceso fuera inverso al de la encriptación.

  Descompresión por RLE:

  Se desarrolló primero este método por su simplicidad, permitiendo validar más rápido la lógica de prueba de parámetros.

  Descompresión por LZ78:

  Se añadió después, con un diccionario dinámico de hasta 65536 entradas, cumpliendo con la especificación del desafío.

  Integración en la búsqueda de parámetros:

  Se creó la función encontrarParametrosYMetodo, que prueba todas las combinaciones y valida con la pista para identificar el método correcto.

Optimización y pruebas finales:

Se optimizó la liberación de memoria y se verificó la correcta reconstrucción del mensaje original con ejemplos de prueba.