Informe Desafío 1

Informática 2

Alejandro Bedoya Zuluaga
Jerónimo Espinosa Herrera

Universidad de Antioquia - Facultad de Ingeniería
28 de septiembre de 2025

1. Introducción

Este proyecto implementa un programa que puede recuperar mensajes que han sido comprimidos y después encriptados. El programa usa fuerza bruta para probar diferentes combinaciones de parámetros de desencriptación hasta encontrar la correcta, verificando el resultado con una pista proporcionada.

2. Descripción del Problema

Un mensaje de texto pasa por dos procesos consecutivos:

Compresión: Se usa RLE (Run-Length Encoding) o LZ78. RLE convierte secuencias repetidas como .^AAAABBB.^en "4A3B". LZ78 crea un diccionario de subcadenas y las referencia con números.

Encriptación: Se aplica XOR con una clave específica y después se rotan los bits de cada byte hacia la derecha.

El objetivo es encontrar los parámetros correctos de desencriptación y el algoritmo de compresión usado.

"'latex

3. Análisis del enfoque adoptado

Para abordar la resolución de este problema se optó por un enfoque de tipo *fuerza bruta*, procurando en todo momento no sacrificar la eficiencia ni el uso adecuado de la memoria dinámica. Esta decisión se fundamenta en la gran cantidad de posibles combinaciones de procesos de compresión y encriptación que pueden haberse aplicado al texto original.

De manera general, la idea de la solución puede resumirse en los siguientes pasos:

1. Leer los archivos `.txt` correspondientes al texto comprimido y encriptado, junto con el archivo que contiene la pista.
2. Aplicar los distintos métodos de desencriptado y descompresión sobre el archivo de entrada.
3. Realizar, en paralelo, la búsqueda de coincidencias con la pista proporcionada.
4. Una vez identificados los procesos utilizados, almacenar los resultados obtenidos en un archivo de salida en formato `.txt`.

Para llevar a cabo esta solución se definieron diversas funciones, organizadas en módulos independientes con el propósito de favorecer la claridad, la modularidad y la mantenibilidad del código desarrollado. "'

4. Estructura del Código

4.1. Módulo ManipulacionTexto

Función leerArchivoACharArray:

```
unsigned char* leerArchivoACharArray(const char* rutaArchivo, int& size)
```

Abre un archivo en modo binario, determina su tamaño y reserva memoria dinámica para cargar todo el contenido. Añade un terminador nulo al final y retorna un puntero al contenido o nullptr si hay error.

Función crearArchivoConTexto:

```
bool crearArchivoConTexto(const char* rutaArchivo, unsigned char* texto,
    int size)
```

Crea un archivo nuevo y escribe en él exactamente los bytes recibidos en modo binario. Retorna true si tiene éxito, false si no puede crear o escribir el archivo.

Función mostrarContenido:

```
void mostrarContenido(unsigned char* contenido, int size)
```

Recorre el arreglo byte por byte. Si encuentra letras minúsculas (a-z) las muestra como caracteres, el resto las muestra como números enteros.

4.2. Módulo CompresionDescompresion

Función descompresionRLE:

```
unsigned char* descompresionRLE(unsigned char* data, int size,
                                unsigned char claveXOR, int rotacionBits,
                                int& total, bool& esValido)
```

Procesa los datos en grupos de 3 bytes (ternas). Para cada terna aplica XOR con la clave dada, después rota los bits hacia la derecha. Interpreta el segundo byte como número de repeticiones y el tercer byte como carácter a repetir.

Hace dos pasadas: primera valida los datos y calcula el tamaño total necesario, segunda construye el resultado. Si encuentra caracteres inválidos o números de repetición incorrectos, establece esValido en false y retorna nullptr.

Función descompresionLZ78:

```
unsigned char* descompresionLZ78(unsigned char* data, int size,
                                 unsigned char claveXOR, int rotacionBits
    ,
                                 int& total, bool& esValido)
```

También procesa en ternas aplicando XOR y rotación. Los primeros dos bytes forman un número que referencia una entrada del diccionario, el tercer byte es el carácter nuevo.

Construye un diccionario dinámico. Para cada terna crea una nueva entrada combinando la cadena referenciada con el carácter nuevo. Si la referencia es 0, solo usa el carácter nuevo. Hace dos pasadas como RLE.

4.3. Módulo BusquedaParametros

Función buscarSecuencia:

```
bool buscarSecuencia(unsigned char* texto, int sizeTexto,
                     unsigned char* pista, int sizePista)
```

Implementa búsqueda de patrones simple. Para cada posición posible en el texto compara byte por byte con la pista. Retorna true si encuentra coincidencia completa.

Función ProbarDescompresion:

```
bool ProbarDescompresion(unsigned char* data, int size,
                         unsigned char* pista, int sizePista,
                         unsigned char claveXOR, int rotacionBits,
                         const char* rutaArchivoModificado)
```

Coordina el proceso de probar una combinación específica de parámetros. Primero intenta RLE, si funciona y encuentra la pista, guarda el archivo y retorna true. Si no, intenta LZ78. Muestra información cuando encuentra la combinación correcta.

Función BuscarParametros:

```
bool BuscarParametros(unsigned char* data, int& nbits, int& claveK,
                      int sizeEncriptado, int sizePista,
                      unsigned char* pista,
                      const char* rutaArchivoModificado)
```

Implementa la búsqueda exhaustiva con doble bucle: claves XOR de 0 a 254 y rotaciones de 0 a 7. Para cada combinación llama a ProbarDescompresion. Se detiene cuando encuentra la combinación correcta. Muestra progreso cada 200 intentos y tiene límite máximo de 2,040 intentos.

4.4. Función Principal

Función main:

```
int main()
```

Procesa múltiples archivos en secuencia. Construye automáticamente los nombres de archivos cambiando un dígito en las rutas. Para cada archivo lee tanto el encriptado como la pista, llama a BuscarParametros, y libera la memoria. Continúa con el siguiente archivo aunque haya errores en el actual, esta funcion luego llama a la funcion BuscarParametros para asi iniciar el flujo del programa y empezar con la busqueda de "fuerza bruta"probando las posibles combinaciones

5. Algoritmos Implementados

El programa usa fuerza bruta para probar 2,040 combinaciones posibles (255 claves XOR $\times$ 8 rotaciones). Para cada combinación:

1. Aplica desencriptación integrada durante la descompresión 2. Valida el formato de los datos antes de procesar completamente 3. Busca la pista en el resultado 4. Si encuentra coincidencia, guarda el archivo y termina la búsqueda

Las funciones de descompresión integran XOR y rotación durante el procesamiento, evitando crear copias adicionales de los datos. La validación temprana permite detener el procesamiento tan pronto como se detecta que los parámetros son incorrectos.

6. Funcionamiento del Programa

El programa lee archivos encriptados y sus pistas correspondientes. Prueba sistemáticamente todas las combinaciones de parámetros de desencriptación. Cuando encuentra la combinación que produce un texto conteniendo la pista, guarda automáticamente el resultado y reporta los parámetros utilizados.

El sistema está diseñado para procesar múltiples archivos de forma automática y continua funcionando aunque encuentre errores en archivos individuales. “‘latex

7. Conclusiones

Aunque la solución propuesta se basa principalmente en un proceso de *fuerza bruta*, se procuró optimizar en la medida de lo posible la eficiencia del programa mediante un uso responsable de la memoria dinámica y la reducción de operaciones innecesarias. Para ello se implementaron recorridos estructurados en ternas, se minimizaron iteraciones redundantes y se evitó la creación de funciones superfluas.

Asimismo, se incorporaron validaciones tempranas para identificar datos inválidos, lo cual permite detener oportunamente la ejecución de procesos inconsistentes y evita que el programa entre en ciclos de ejecución innecesarios.

El desarrollo de este proyecto resultó de gran utilidad para fortalecer nuestra capacidad de análisis frente a desafíos con mayor cercanía a problemas reales. Además, permitió poner en práctica y comparar diferentes estrategias algorítmicas, generando un aprendizaje significativo tanto en términos de diseño de soluciones como en el uso eficiente de recursos en C++. “‘