INFORME DESAFIO 1

INFORMATICA II

Cristian Camilo Murillo Jiménez

C.C 1017926405

Juan Felipe Pérez Salazar

C.C1017922201

UNIVERSIDAD DE ANTIOQUIA

2025

Durante el análisis inicial se concluyó que el problema podía abordarse desde la perspectiva de la **ingeniería inversa**. A través de este enfoque, fue posible reconstruir el mensaje codificado, pasando por el proceso de decodificación y descompresión del texto para, posteriormente, buscar la pista solicitada.

En un primer momento se consideró la posibilidad de **codificar la pista** y buscarla directamente dentro del mensaje comprimido. Sin embargo, se identificó que al no encontrarse necesariamente al inicio del texto, métodos como la compresión **LZ78** podían generar conflictos en la solución. Por tal motivo, se optó por una estrategia diferente que consistió en crear una ventana de búsqueda en el texto, empleando una lógica similar a la utilizada en algoritmos para la detección de palíndromos.

Con base en este análisis, el problema se dividió en **tres bloques principales**:

1. **Lectura del archivo**
   En este bloque se desarrollaron funciones destinadas a la correcta manipulación de archivos de entrada y su almacenamiento en memoria:
   - char *leerArchivo(const char* nombreArchivo, long longitud);
   - long obtenerLongitudArchivo(const char *nombreArchivo);
   - void intAChars(int num, char* buffer, int& pos);
   - char* construirNombre(const char* base, int num, const char* extension);

Cada función cumplió un rol específico, desde la lectura del archivo, el cálculo de su longitud y la construcción dinámica de nombres de archivo, hasta la conversión de números a cadenas de caracteres.

2. **Decodificación y descompresión del texto**
   Este bloque tuvo como objetivo principal transformar el mensaje encriptado en su forma legible. Para ello, se implementaron las siguientes funciones:
   - unsigned char* j_desencriptarMensaje(unsigned char* j_dataEncriptada, unsigned int j_tamano, int j_claveK, int j_rotacionN);
   - char* convertirABin(int numero);
   - void quitarEspaciosEnMemoria(unsigned char* datos);

Posteriormente, se desarrollaron las funciones correspondientes a la **descompresión RLE**:

   - unsigned char* j_descomprimirRLE(unsigned char* j_dataComprimida, unsigned int j_tamComprimido, unsigned int& j_tamDescomprimido);

- unsigned char* j_comprimirRLE(unsigned char* j_dataOriginal, unsigned int j_tamOriginal, unsigned int& j_tamComprimido);

Cabe resaltar que, en la implementación final, la descompresión RLE fue la única que funcionó correctamente. No obstante, esta presenta una limitación: el número máximo de repeticiones de un carácter permitido es 9, dado que el algoritmo solo admite un dígito para representar la cantidad.

3. **Búsqueda de la pista**
   Una vez decodificado y descomprimido el mensaje, se desarrolló la función:

   int encontrarPista(const unsigned char* textoOriginal, const char* textoPista);

Esta función permitió realizar la búsqueda de la pista dentro del mensaje reconstruido, aplicando la lógica definida en la fase de análisis.

Problemas identificados y soluciones adoptadas

El primer inconveniente detectado se relacionó con la compresión **LZ78**. Debido a que la pista no se encontraba garantizada al inicio del mensaje, no fue posible implementar este método de manera adecuada, lo cual obligó a replantear la estrategia inicial.

En la implementación final, el principal problema estuvo asociado al **uso intensivo de memoria**. La gran cantidad de operaciones realizadas provocaba que el programa saturara la memoria y abortara la ejecución. Este inconveniente se solucionó mediante una validación previa, evitando que el proceso completo se ejecutara cuando el mensaje de entrada resultaba inválido.