Informe de Desarrollo del Proyecto – Desafío de Descompresión y Desencriptado

a. Análisis del problema y consideraciones para la alternativa de solución propuesta

El problema central consistió en recuperar el contenido original de varios archivos que habían sido sometidos a un proceso de compresión (algoritmos RLE y LZ78 en variantes específicas) y luego encriptados mediante un esquema sencillo basado en rotación de bits y operación XOR.

Las consideraciones principales fueron:
- Restricciones del entorno: no se permitió usar bibliotecas de alto nivel como STL, ni funciones estándar para tratar datos de tipo string.
- Tamaño de los archivos: dado que en el enunciado del desafío no se especificó el tamaño que los archivos podrían llegar a tener, se escogió 'unsigned long' como tipo de índice y Contador para así no tener problemas en textos muy grandes.
- Formato ambiguo: el método RLE se presentó en más de una variante (ASCII y ternas con byte nulo). Se decidió implementar la variante RLE_TERNA como formato principal, al ser la más consistente con las evidencias de los archivos.
- Seguridad de memoria: al no usar STL, fue necesario manejar manualmente punteros, asignación y liberación de memoria.

La solución propuesta fue un programa en C++ capaz de:
1. Leer los archivos binarios encriptados.
2. Aplicar sistemáticamente todas las combinaciones de parámetros (n, K) para desencriptar.
3. Intentar la descompresión con los algoritmos definidos.
4. Validar el resultado mediante una pista provista en un archivo auxiliar.
5. Generar un reporte con el texto original y los parámetros hallados.

b. Esquema de tareas definidas en el desarrollo de los algoritmos

1. Lectura de archivos:
   - Implementar funciones con fstream para leer en modo binario y texto.
2. Funciones auxiliares:
   - Implementación de 'contar_chars', 'comparar_n_chars', 'memcpy'.
   - Conversión de números a cadenas (utoa_dec) y a hexadecimal (byte_to_hex).

3. Desencriptado:
  - Función que aplica XOR con la llave K y luego rotación a la derecha n bits.
4. Compresión / descompresión:
  - 'decompress_rle_terna': descompresión por ternas [0x00][count][symbol].
  - 'decompress_lz78': reconstrucción de frases a partir de tripletas (idx, char) con diccionario dinámico.
5. Validación de la pista:
  - Implementación de 'buscar_pista'.
6. Pruebas exhaustivas:
  - Ciclos anidados por todos los valores de n (1–7) y K (0–255).
7. Reporte final:
  - Generación de SalidaX.txt con método, parámetros y texto completo.

c. Algoritmos implementados

- Desencriptado XOR+ROR: dec[i] = ROR(enc[i] ^ K, n).
- RLE TERNA:
  Formato: [0x00][count][symbol] repetido.
  Cada terna genera count repeticiones del symbol.
- LZ78 (variante simple):
  Tripletas [prefijo_hi][prefijo_lo][char], con índice 1-based, big-endian.
  Diccionario dinámico: cada nueva frase se forma concatenando dict[idx] + char.
  Salida = concatenación de todas las frases en orden.

d. Problemas de desarrollo afrontados

- Confusión en el formato de RLE: inicialmente se intentó con RLE ASCII, pero no funcionó en ninguno de los casos. Fue necesario implementar la variante RLE TERNA, después de darse por aclarado el formato por el docente del curso.

-Complejidad en la implementación del algoritmo lz78: El algoritmo lz78 presentó varios retos a la hora de implementar una solución efectiva y eficiente en código porque no se tenía claro cómo funcionaba el formato de dos bytes de posición y cómo interpretar eso como un único valor, pero después de buscar información respecto a como se podría solucionar, se tuvo que aplicar el formato de ordenamiento de dos bytes BIG ENDIAN para poder interpretarlo como un único valor.

- Se presentó cierta dificultad a la hora de manejar las estructuras propias de los punteros dobles, puesto que no se tenía completa transparencia del funcionamiento interno de los mismos por lo que fue difícil comprender y a la vez implementar una estructura que usara este tipo de variables esenciales en este tipo de retos.

- Desconcierto respecto a cómo leer los archivos de los datasets de Desarrollo de modo que se pudiera distinguir correctamente sobre terna de bytes, ya que en algunas ocasiones los bytes vacíos al inicio de cada terna no tienen sentido dada la forma en la que los archivos deberían verse [número][letra].

- Manejo de memoria: al no usar STL, se tuvieron que implementar rutinas de liberación cuidadosa (delete[]) en cada fase.
- Conflictos de integración: durante el versionado en Git, se presentaron conflictos al mezclar versiones del archivo main.cpp con diferentes implementaciones de RLE.

e. Evolución de la solución y consideraciones para la implementación

- Se inició con funciones básicas de manipulación de cadenas y copia de memoria.

- Se implementó primero un descompresor RLE en ASCII para luego finalmente implementar RLE_TERNA.

- Posteriormente se agregó LZ78 con diccionario dinámico basado en punteros dobles.

- Se refinó el proceso de desencriptado hasta asegurar que los parámetros (n, K) correctos podían encontrarse mediante búsqueda exhaustiva.

- Se agregaron funciones únicas para hacer el reporte de salida de todos los archivos analizados, tal y como lo indican los datasets de desarrollo.

- La solución evolucionó de ser un prototipo dependiente de STL a un programa autocontenido, ajustado a las restricciones del desafío.