**Informe Preliminar Desafío 1**

**Integrantes:** Sara Echeverri Giraldo, Luis Miguel Alzate

**Curso:** Informática II

**Introducción**

Este desafío consiste en resolver un problema de ingeniería inversa aplicado a textos comprimidos y encriptados, se trabajará en lenguaje C++ con arreglos dinámicos y punteros, y se gestionará en git para evidenciar el proceso de construcción paso a paso.

**Contextualización del problema**

Básicamente, se nos entrega un archivo .txt que contiene un texto primero comprimido (mediante un algoritmo RLE o LZ78) y posteriormente encriptado usando dos operaciones a nivel de bits: una rotación de n bits hacia la izquierda y un XoR con una clave K.

Adicionalmente, se proporciona un fragmento conocido del texto original en claro, que funciona como pista para identificar los parámetros de encriptación y el método de compresión.

El reto consiste en:

1. Descubrir el valor de n (cuántos bits fueron rotados).
2. Descubrir el valor de K (la clave del XoR).
3. Determinar qué método de compresión se usó (RLE o LZ78).
4. Desencriptar y descomprimir el archivo completo, obteniendo nuevamente el texto original.

### **Análisis de solución**

Para deshacer la encriptación se aplica el orden inverso al que se utilizó al cifrar. El esquema de encriptación consistía en dos pasos:

1. Rotación circular a la izquierda de cada byte n posiciones.
2. Aplicación de un XOR bit a bit con la clave K.

Dado que el XOR es su propia inversa, en el proceso de desencriptación se aplica nuevamente el XOR con la misma clave K. Posteriormente, se invierte la rotación: se aplica una rotación circular a la derecha de n posiciones.

En resumen:

* **Encriptar**: rotarIzquierda (byte, n) —> XoR
* **Desencriptar**: XoR —> rotarDerecha(byte, n)

### 

### **Estrategia para encontrar *n* y *K***

Una estrategia directa consiste en probar todas las combinaciones de n (7 valores posibles) y K (256 valores posibles) sobre el fragmento comprimido conocido, y verificar si el resultado coincide con una porción del ciphertext. Este método garantiza encontrar la clave, pero puede resultar poco eficiente (1792 combinaciones por cada fragmento).

Existe, sin embargo, un enfoque determinista más eficiente:

* Si se toman dos posiciones distintas del ciphertext (C[i] y C[j]), y se las combina mediante XOR, la clave K desaparece de la ecuación:

C[i] + C[j] = rotl(S[i] , n) + rotl(S[j] , n) = rolt(S[i] + S[j] , n)

* De esta forma, solo es necesario probar los 7 valores de n hasta encontrar aquel que alinea las diferencias entre el fragmento comprimido esperado (S) y el ciphertext (C).
* Una vez encontrado n, el valor de K se calcula de inmediato con un solo byte:

K = rolt(S[0] , n) + C[i]

Este enfoque reduce drásticamente el espacio de búsqueda y garantiza una detección más rápida y precisa de los parámetros de encriptación.

**Diseño de la solución**

Con el fin de abordar de manera ordenada el desafío, se plantea un diseño modular inicial que permita dividir el trabajo en componentes bien definidos, asegurar la eficiencia en el manejo de memoria y facilitar la integración final.

**1. Módulo de encriptación y desencriptación**

* Funciones principales:
  + rotderecha: realizan la rotación de bits a la derecha en cada byte, controlada por el parámetro *n*.
  + xor\_buffer: aplica la operación XOR a cada byte del arreglo con la clave *K*.
* Lógica: dado que la encriptación original fue *rotación izquierda + XOR*, la desencriptación se aplicará en orden inverso: primero XOR con *K* y luego rotación a la derecha con *n*.

**2. Módulo de compresión y descompresión**

* RLE (Run-Length Encoding):
  + rle\_compresión y rle\_descompresión.
  + El método de compresión convierte secuencias repetidas en números ASCII seguidos del símbolo.
  + La descompresión reconstruye la secuencia expandiendo cada conteo.
* LZ78:
  + lz78\_compresión y lz78\_descompresión.
  + Utiliza un diccionario dinámico de hasta 65.536 entradas (índice de 2 bytes) y un carácter final.
  + La descompresión reconstruye el texto original concatenando el prefijo referenciado con el nuevo carácter.
* Validaciones: en ambos métodos se controlará la coherencia de los datos y la asignación dinámica de memoria mediante malloc, realloc y free.

**3. Módulo principal**

* Flujo general:
  1. Cargar el archivo encriptado en memoria.
  2. Comprimir el fragmento conocido con RLE y LZ78.
  3. Para cada combinación posible de *n* (1 a 7) y *K* (0 a 255):
     + Aplicar la encriptación simulada (rotación izquierda + XOR) sobre el fragmento comprimido.
  4. Al encontrar coincidencia:
     + Desencriptar el archivo completo.
     + Descomprimir con el método correcto (RLE o LZ78).
     + Guardar el texto original recuperado.

**4. Gestión de memoria**

* Se emplearán exclusivamente punteros y arreglos dinámicos, sin uso de std::string ni contenedores de la STL.
* Se validarán los límites de memoria y se liberarán todos los recursos cuando no se necesiten más.

**5. Validación y pruebas**

* Se desarrollarán pruebas unitarias para cada módulo (RLE, LZ78, rotación, XOR).
* Se verificará que el fragmento conocido aparezca correctamente en la salida final, confirmando que los parámetros hallados son los correctos.

**6. Enfoques descartados**

* El primer enfoque que tuvimos en general fue hacer las operaciones de encriptación en el fragmento de prueba y de ahí compararlo con el fragmento encriptado entero, así reduciendo el volumen de datos para poder hacer las pruebas con mucho menos uso de la memoria que haciendo todo directamente con el texto grande. El problema con este enfoque era la rotación de bits, ya que, al ser una rotación cuadrada, trabajar con cantidades de datos tan diferentes haría que los resultados de esta operación no fueran equivalentes

**Conclusiones del análisis inicial**

* El problema nos obliga a combinar conocimientos de compresión, encriptación de bajo nivel y búsqueda de patrones.
* La solución propuesta es:
  1. Generar la pista comprimida con ambos métodos.
  2. Simular la encriptación con todos los n y K posibles.
  3. Buscar coincidencias en el archivo encriptado.
  4. Con los parámetros hallados, desencriptar y descomprimir todo.

Este enfoque asegura eficiencia, permite trabajar en equipo (dividiendo funciones de compresión, encriptación y búsqueda) y cumple con las restricciones del desafío (uso de arreglos dinámicos, sin STL, con punteros y memoria controlada).