**Preinforme desafío I**

**Duvan Camilo Aragón Gallego**

**TI. 1033183697**

**David Fernando Revelo Morales**

**C.C 1085909373**

**Informática II**

**Aníbal Jose Guerra Soler**

**Universidad de Antioquia**

**Medellín**

**2025**

El desafío es un ejercicio de ingeniería inversa que requiere reconstruir un mensaje original que ha sido sometido a dos procesos secuenciales: compresión y encriptación. La única información que nos proporciona el cliente es un mensaje encriptado y comprimido, junto con un pequeño fragmento del texto original en texto plano. El objetivo principal es identificar el método de compresión (RLE o LZ78), los parámetros de encriptación (rotación de bits n y clave k), y luego utilizar esta información para desencriptar y descomprimir el mensaje completo. Para intentar hallar los valores correctos de k y n podemos partir de que la manera más simple pero compleja en termino de recursos es empezar a evaluar todos los valores de k y n hasta llegar a los valores que funcionan, pero esto requiere muchas búsquedas innecesarias, para intentar optimizar un poco esto, vamos a hacerlo de la siguiente manera:

1. **Búsqueda de la clave y cantidad de bits rotados:**

Sabemos que:

* **k** (la clave) es un valor entre 0 y 255.
* **n** (cantidad de bits a rotar hacia la derecha para desencriptar) está entre 1 y 7.

Esto significa que, en teoría, habría que probar hasta 1785 combinaciones posibles para cada carácter. Para no tener que recorrer todas esas combinaciones de manera exhaustiva, aplicamos un enfoque que permite descartar candidatos inválidos rápidamente.

El procedimiento que pensamos para lograr esto es el siguiente:

1. Seleccionamos el valor inicial de k en 0.
2. Tomamos el valor ASCII del primer carácter y aplicamos la operación XOR de ese valor por k.
3. Se aplican rotaciones de bits (1-7 posiciones) al valor obtenido del paso anterior.
4. Verificamos si cada uno de los valores resultantes están dentro de los rangos permitidos en la tabla ASCII sabiendo que el texto solo puede incluir:

* Letras minúsculas: a-z.
* Letras mayúsculas: A-Z.
* Dígitos: 0-9.
* En caso de codificación LZ78 también se permiten: ( , ) y , .

1. Si el carácter transformado está dentro del rango, registramos ese valor de n como candidato válido.
2. Repetimos el proceso con el siguiente carácter del texto, pero solo comprobamos los valores de n que funcionaron en el paso anterior.

* Esto se debe a que tanto k como n deben ser los mismos para todo el texto.

1. Si en algún momento un valor de n deja de generar caracteres válidos, lo descartamos.
2. Si llega un momento donde no quede ningún valor de n, descartamos ese k y pasamos al siguiente valor de k.

Con este método, aunque en el peor de los casos se debe seguir probando las 1785 combinaciones, en la práctica se reduce mucho las pruebas porque los candidatos inválidos se descartan muy pronto.

1. **Validación de compresión usada y búsqueda de pista:**

Una vez encontramos un texto candidato válido (es decir, descifrado complemento con cierto (k, n)) debemos comprobar si contiene la pista dada para ello lo hacemos de la siguiente manera:

1. **Intento con RLE**

* Comprimimos el texto de pista usando el algoritmo RLE.
* Revisamos si encontramos el fragmento exacto en el texto comprimido.
* Si la pista comprimida existe, hemos encontrado la solución correcta: (k, n) y el texto descifrado después de descomprimir.

1. **Intento con LZ78**

* Si la contraseña no aparece en RLE, comprimimos la misma pista, pero en LZ78.
* Revisamos nuevamente si encontramos el fragmento exacto en el texto comprimido.
* Si la pista comprimida existe, hemos encontrado la solución correcta: (k, n) y el texto descifrado después de descomprimir.

1. **Si no aparece en ninguno de los dos casos**

* Descartamos ese (k, n) como solución.
* Continuamos con el siguiente valor de **k** y repetimos el procedimiento.
* Después de buscar en todos los (k, n) si no encontramos ninguno valor válido, no está en un tipo de codificación válida.