Desafío I – Descompresión y Desencriptación

Cristian Florez Monsalve

Thomas Mejia Moncada

Grupo 6

Informática II

Aníbal Jose Guerra Soler

Universidad de Antioquia

Colombia-Medellín

18/09/2025

Análisis del Problema

Un mensaje encriptado es entregado en forma de un archivo de texto plano, mensaje producto de una serie de pasos de compresión y desplazamiento de bits, el mensaje encriptado y comprimido se compone de letras alfabéticas mayúsculas, minúsculas y de números de cero al nueve. El mensaje primero fue comprimido usando alguno de estos dos métodos LRE o LZ78 posterior a comprimir el mensaje se encripto aplicando una máscara de un byte desconocida denominada como K, posterior a eso se camuflo aun mas el mensaje usando desplazamiento de bits a la *Izquierda* una cantidad de n veces. Adicional recibimos otro documento de texto plano que incluye un fragmento de lo que fue el texto original.

El objetivo es aplicar de forma inversa la desencriptación de forma tal que se logre conocer el valor de los n bits que se desplazaron a la izquierda, encontrar la llave k que fue usada a la hora de enmascarar el mensaje, y descubrir cuál de los métodos de compresión fue utilizado a la hora de comprimir el mensaje ya sea LRE o LZ78, una vez hechos todos los pasos se debe verificar que el fragmento entregado como pista corresponda efectivamente a una fracción del texto desencriptado y descomprimido.

información de entrada:

- **Archivo .txt:** Contiene el mensaje encriptado y comprimido.
- **Archivo .txt:** Contiene el fragmento del mensaje original.

Información de salida:

- **Archivo .txt:** Contiene el mensaje original después de desencriptarlo.

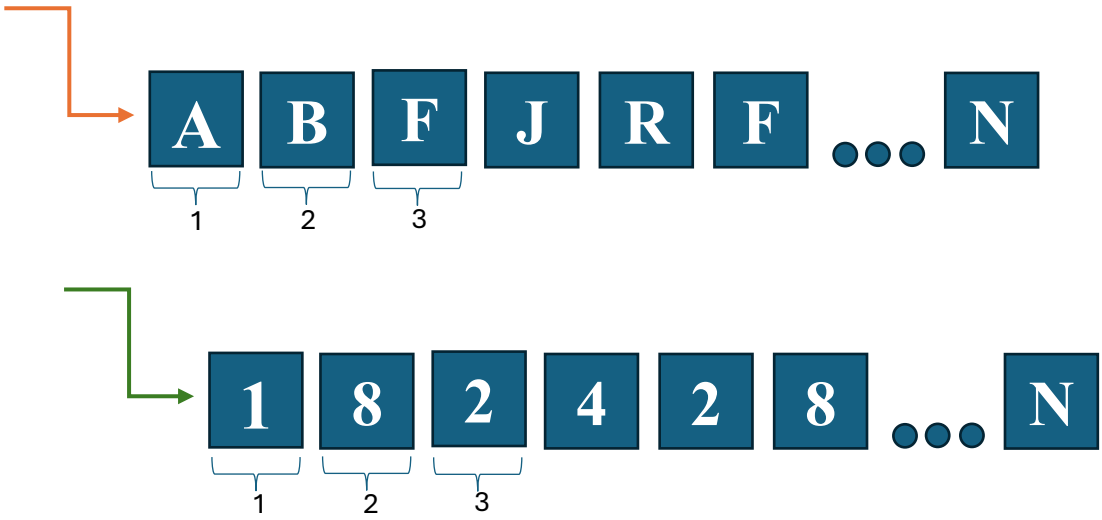
Consideraciones a tener en cuenta:

1. Se deben usar 2 bytes para almacenar de forma correcta y evitar desbordes de bits a la hora de descomprimir los prefijos.
2. Debe incluir un uso adecuado y consciente de la memoria dinámica y el uso de estructuras de datos complejas o simplificadas según lo requiera la solución.

Primer Propuesta: Desarrollo y Solución

Utilizamos un algoritmo de fuerza bruta para buscar un resultado funcional, aunque costoso en el uso de recursos por el alto costo en procesamiento lo que se traduce en mayor tiempo de ejecución.

Estructura de datos: Usaremos 2 arreglos unidimensionales dinámicos desacoplados para hacer un uso sencillo y rápido de los índices y prefijos del texto encriptado, de tal forma que un arreglo corresponde a los datos de tipo carácter que hacen parte del texto, y un arreglo a datos del tipo entero que corresponderán a los prefijos que acompañan los caracteres. Ambos arreglos, aunque desacoplados siguen un orden de índice, donde el elemento del n-esimo índice del primer arreglo corresponde con el elemento del n-esimo índice del segundo arreglo.



Para llevar acabo la solución del problema seguiremos un algoritmo simple que actúa de forma secuencial, así:

1. Leemos el archivo donde se encuentra el texto encriptado y comprimido.
2. Cargamos la información en la estructura de datos.
3. Usamos un ciclo para iterar sobre cada componente que hace parte de la estructura de datos.
4. Luego usamos otro ciclo para iterar desde 0 a 255 de forma binaria para probar todas las posibles keys K que pudieron haber sido usadas.
5. Usamos XOR sobre la key obtenida y cada elemento del arreglo de caracteres.
6. Aplicamos un desplazamiento a la izquierda o a la derecha según nos sea mas conveniente.
7. Probamos con ambos métodos de descompresión LRE o LZ78
8. Iteramos sobre el texto resultante buscando una conincidencia exacta con la pista (fragmento del texto original) si lo encontramos entonces tomamos los valores de k, n y el método de descompresión que usamos

9. Retornamos al usuario tanto los valores usados para solucionar y el texto original después de descomprimir y desencriptar
10. Si no coincide se repite el proceso hasta hallar un k, n y el método de descompresión que fue usado.

Métodos y Funcionalidades Implementadas.

La solución al desafío fue alcanzada gracias a la división del problema en subproblemas para los cuales se programo una tarea especifica donde juntos conforman una de muchas posibles soluciones al problema planteado.

1. Modulo (ReadWriteLoad): Utilizado para la manipulación de archivos de texto plano, con funcionalidades de:
    a. Lectura de archivos (ReadArchivo).
    b. Escritura de archivos (WriteArchivo).
2. Modulo (Verificaciones): Utilizado para la verificación de datos, como la existencia de cadenas dentro de otras cadenas, la existencia de caracteres válidos, y poder hacer testeos rápidos por consola.
    a. Verificación de caracteres válidos (esCaracterInvalido)
    b. verificación de existencia de sub-arreglos dentro de arreglos (in)
3. Modulo (ModifyStruct): Utilizado para separar y dar tratamiento individual a los datos dependiendo el método de descompresión separando los datos de un arreglo en dos arreglos desacoplados.
    a. Separa en dos arreglos teniendo en cuenta 2 de los 3 bytes (SepararArrLRE)
    b. Separa en dos arreglos teniendo en cuenta los 3 bytes (SepararArrLZ78)
4. Modulo (Desencriptado): Utilizado para aplicar las operaciones a nivel de bits y dar tratamiento al mensaje encriptado de un arreglo bit a bit.
    a. Aplica XOR entre un k y un bit (aplicaron)
    b. Aplica una rotación a la derecha de un bit n veces (rotarDerecha)
    c. Desencripta el texto aplicando las 2 funciones anteriores (desencriptarTexto)
5. Modulo (MetodoLZ78): Utilizado para tratar los bytes de los arreglos desacoplados como tuplas índice-prefijo
6. Modulo (MetodoLRE): Utilizado para dar tratamiento a los bits de los arreglos desacoplados como tuplas de cantidad-Identificador.

Métodos de Eficiencia.

Para representar la eficiencia del código en los diferentes puntos del desarrollo tenemos la siguiente tabla que evalúa el tiempo promedio de ejecución en los 4 casos de prueba haciendo uso de la librería chrono para el uso de la funcionalidad que permite medir el tiempo, la primera columna corresponde a la cantidad de caracteres promedio que se tiene en todos los archivos de casos de prueba y una columna enfocada a resaltar la versión y el cambio que hizo que mejorara su tiempo de ejecución. (el tiempo esta medido en segundos).

Promedio Caracteres	Promedio Tiempo Ejecución	Versión	Descripción de Mejora
1398	0.051818625000	0	N/N
1398	0.00512057500	1	Implementamos un condicional que evalúa si el primer carácter el arreglo a descomprimir corresponde a un 0 u otro numero para saber si es LRE o LZ78
1398	0.002895200000	2	Implementamos una función encargada de verificar si en realidad hay un carácter valido en la primera posición como números o letras. Y así evitamos construir estructuras innecesarias

Problemas de Desarrollo

1. Tuvimos problema para hallar la cantidad de memoria necesaria para reservar el arreglo resultante al descomprimir con LZ78, pues a la hora de construir el arreglo requeríamos de una estructura auxiliar para reconstruir el diccionario por ende optamos por un apuntador de apuntadores que guardaran los índices del texto de tal forma que cada apuntador en el arreglo representaba un índice del diccionario para reconstruir el mensaje.
2. Tuvimos problema con la lectura del archivo, por un error de semántica abríamos el archivo de forma errónea con culpa de un carácter reservado que se malinterpreto en la documentación de la librería.
3. Tuvimos problema a la hora de verificar si las subcadenas de la pista estaba dentro de la cadena de caracteres del arreglo desencriptado y descomprimido, pues desechaba posiciones donde podría haber una similitud parcial a partir de cierto punto.
4. Problemas a la hora de interpretar que manejo darles a los archivos, pues notamos que el primer carácter no correspondía a lo que debería ser la forma descomprimida y luego vimos que venían en ternas donde para LRE solo eran validos 2 bytes y para LZ78 3 bytes lo cual nos hizo tener que cambiar la estructura de como descomprimíamos ambos métodos y darle un tratamiento especial a cada uno.
5. Problemas para decidir donde era una liberación oportuna de la memoria de tal forma que no siempre estuvieran presentes grandes cantidades de datos de forma innecesaria.

Estructura auxiliar (Para el diccionario de LZ78)

Por ejemplo, para el siguiente texto comprimido se crearía el siguiente diccionario auxiliar { (0, A) (0, B) (1, A) (1, B) (4, A) (3, B)…. }

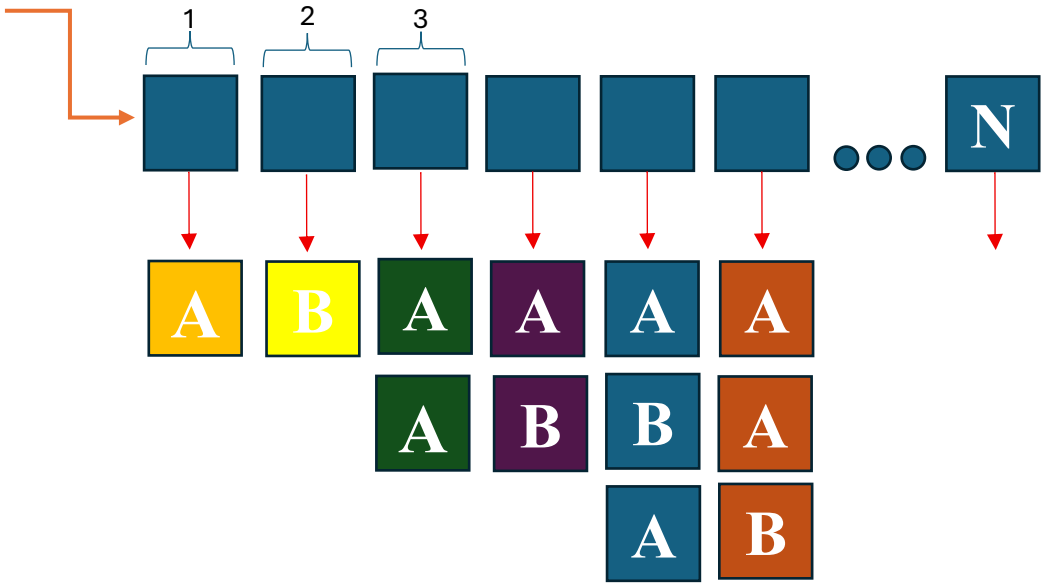


Diagrama de flujo