Universidad Rafael Landívar
Facultad de ingeniería
Ingeniería en informática y sistemas
Estructura de datos II, sección 01
Docente: Ing. Fredy Alexander Bustamante

SISTEMA DE GESTIÓN DE ARCHIVOS
Proyecto 02

Estudiantes:
Balán Mendoza, Tony Alexander - 1202124
González Pérez, Pablo Javier - 1211624
Miranda Abrego, Andrea Sofía - 1065824

Guatemala, 05 de noviembre del 2025

I. INTRODUCCIÓN

El proyecto tiene el objetivo de representar un sistema de gestión de archivos que permita la gestión y almacenamiento seguro y eficiente de archivos de texto mediante la aplicación de algoritmos de compresión y encriptación. Como menciona el enunciado del proyecto, la aplicación surge de la problemática donde una empresa necesita gestionar grandes cantidades de archivos de texto de clientes, los cuales deben estar manejados de manera eficiente en todos los puntos mencionados anteriormente.

El sistema permite trabajar tanto con archivos individuales como con carpetas, leyendo archivo por archivo de manera recursiva. De la misma manera, combina algoritmos de compresión LZ77 Y Huffman para ayudar a optimizar el proceso y usa un sistema de contraseñas para el proceso de encriptación con el fin de proteger mejor los datos.

II. DESCRIPCIÓN DE LA APLICACIÓN

El sistema de gestión de archivos seguros y eficientes en una aplicación de Java realizada para poder gestionar archivos de texto, teniendo la posibilidad de comprimirlos, descomprimirlos, encriptarlos y desencriptarlos. Todo se realiza dentro de un menú de consola y si se desean cambiar los archivos, en la clase menú se actualizan las rutas correspondientes a la entrada y las salidas.

En cuanto a la información general de la funcionalidad, en la compresión se utilizan algoritmos LZ77 y Huffman, mientras que en la encriptación se hace por medio de una contraseña. Así mismo, la aplicación permite recuperar o revertir los archivos (descompresión, desencriptación), permite usar tanto carpetas como archivos individuales y genera un log mostrando los datos básicos de los archivos y su acción determinada. Las extensiones usadas son .cmp para archivos comprimidos, .enc para archivos encriptados, .ec para archivos comprimidos y encriptados y .txt para archivos recuperados.

III. DISEÑO

3.1. ALGORITMOS USADOS

Como se mencionó anteriormente, en la compresión se usó una combinación de los algoritmos LZ77 y Huffman. LZ77 posee una ventana de 4096 bytes y un buffer de búsqueda de 256 bytes. En cuanto a Huffman, se calculan frecuencias de cada token, se construye un árbol, se le asignan códigos de longitud variable y se serializa el árbol. En cuando a la encriptación, en esta se utilizó el algoritmo XOR con clave derivada.

3.2. ARCHIVOS COMPRIMIDOS

Como tal, primero se hace una lectura del archivo de texto, luego se realiza el análisis del texto en la ventana, generando tokens para las secuencias repetidas y caracteres únicos. Luego, dichos tokens se convierten a string y se procede a realizar la codificación de Huffman. En Huffman, se calculan las frecuencias de los tokens, se construye el árbol, se generan códigos y se serializa el árbol. Al finalizar se guarda la información en un archivo .cmp y se almacena en la ruta establecida.

3.3. ARCHIVOS ENCRIPTADOS

Primero se verifica/crea la carpeta de salida (salida). Luego se arma el archivo final con el nombre especificado en NombreSalida. Con esto aseguramos que el resultado vaya exactamente al lugar y con el nombre esperado (por ejemplo, documento.ec o reporte.enc, según decidan). Se abre un FileInputStream sobre entrada. Esto permite leer cualquier contenido byte a byte: texto, binarios, etc. Esto ya que los archivos comprimidos están escritos en bytes. Paralelamente se abre un FileOutputStream hacia el archivo de salida para ir escribiendo el resultado cifrado.

Pasos:

1.Se convierte el carácter a entero (posición dentro de la contraseña).

2.Se calcula clave = p^valorcontra mod g con p = 5 (pequeño) y g = 100000019 (primo grande).

3.Se devuelve un byte con ese valor (con & 0xFF luego, para mantenerlo en 0–255).

4.Mientras leemos bytes del archivo, vamos recorriendo contraseña con indexcontra = (indexcontra + 1) % password.length(). Es decir, si la contraseña

tiene, por ejemplo, 6 caracteres, después del sexto carácter se vuelve al primero, y así sucesivamente.

5.int cifrado = (char)(i ^ claveint) & 0xFF; Aquí tomamos el byte leído (i), lo combina con la clave de turno (claveint) usando XOR y escribe el resultado.

byte a byte, con una “key stream” determinada por la contraseña. Es importante notar que XOR es simétrica: si se vuelve a aplicar el mismo XOR con la misma clave sobre el dato cifrado, se recupera el original.

6.Cada byte cifrado se escribe en el FileOutputStream del archivo de salida. Al final, se cierran ambos streams y se muestra la ruta del archivo encriptado creado. No se borra el original, por lo que quedan ambos archivos: el de entrada sin cambios y el de salida cifrado.

El método Desencriotacion recorre el archivo cifrado exactamente igual, pero al aplicar el mismo XOR con la misma secuencia de claves sobre los bytes cifrados, se obtiene de vuelta el contenido original.

IV. CONCLUSIONES

- Se determinó que la combinación entre LZ77 y Huffman resultó ser efectiva debido a que LZ77 reduce eficientemente la repetición del texto y Huffman termina de optimizar la representación de los tokens.
- Se determinó que la encriptación mediante el algoritmo de XOR proporciona seguridad de manera parcial mientras se asegure correctamente la clave y se mantenga compleja.
- Se determinó que el tiempo de compresión dependerá de tamaño del archivo, llevando a que tasas grandes requieran mayor procesamiento por parte del sistema.
- Se determinó que la serialización de los árboles Huffman ayuda a recuperar los datos para una futura descompresión.