Universidad Rafael Landívar

Facultad de Ingeniería

Ingeniería en Informática Y Sistemas

Estructuras de Datos II

Catedrático: Fredy Bustamante

Proyecto NO.2

Rodrigo José Ruiz Juares-1037623

Índice

Algoritmos utilizados

En este caso se necesitan citar a 2 métodos dentro del programa el primero ProcessPath(…) el segundo recoverPath(…) en este caso el primero se encarga de comprimir y encriptar según la elección se necesite y del recoverPath() se encarga de hacer la recuperación o la inversa de las operaciones.

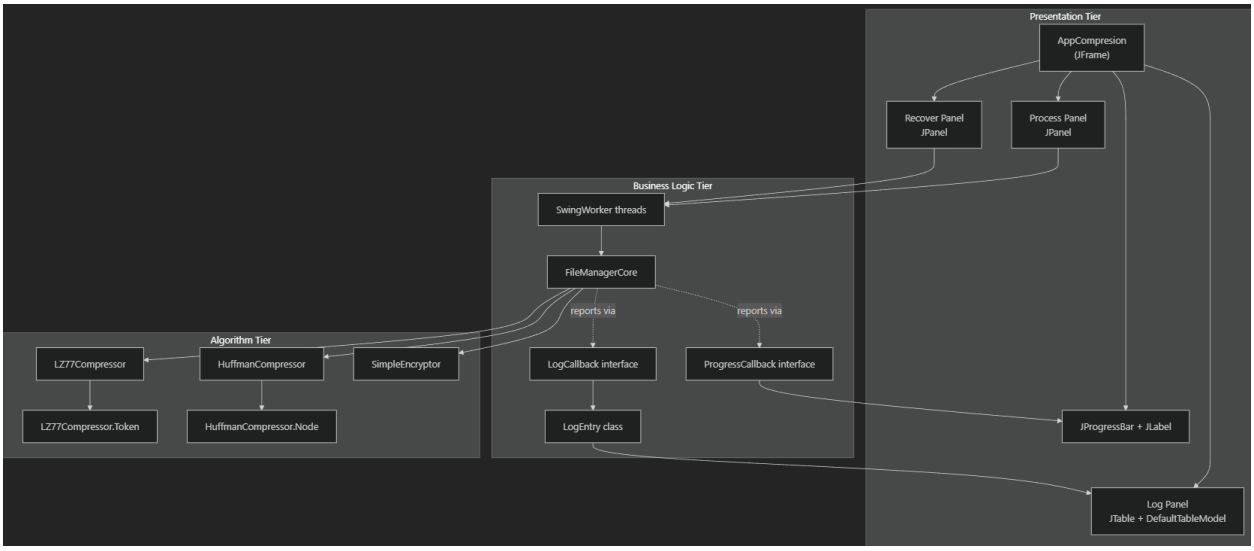
En estos casos se encargan de hacer lo siguiente:

- Detectar si la ruta es archivo o carpeta.
- Recorrer carpetas recursivamente.
- Llamar en orden correcto a los algoritmos (LZ77, Huffman, XOR).
- Medir tiempos de operación y calcular tasa de compresión.
- Reportar progreso y generar entradas en el log.

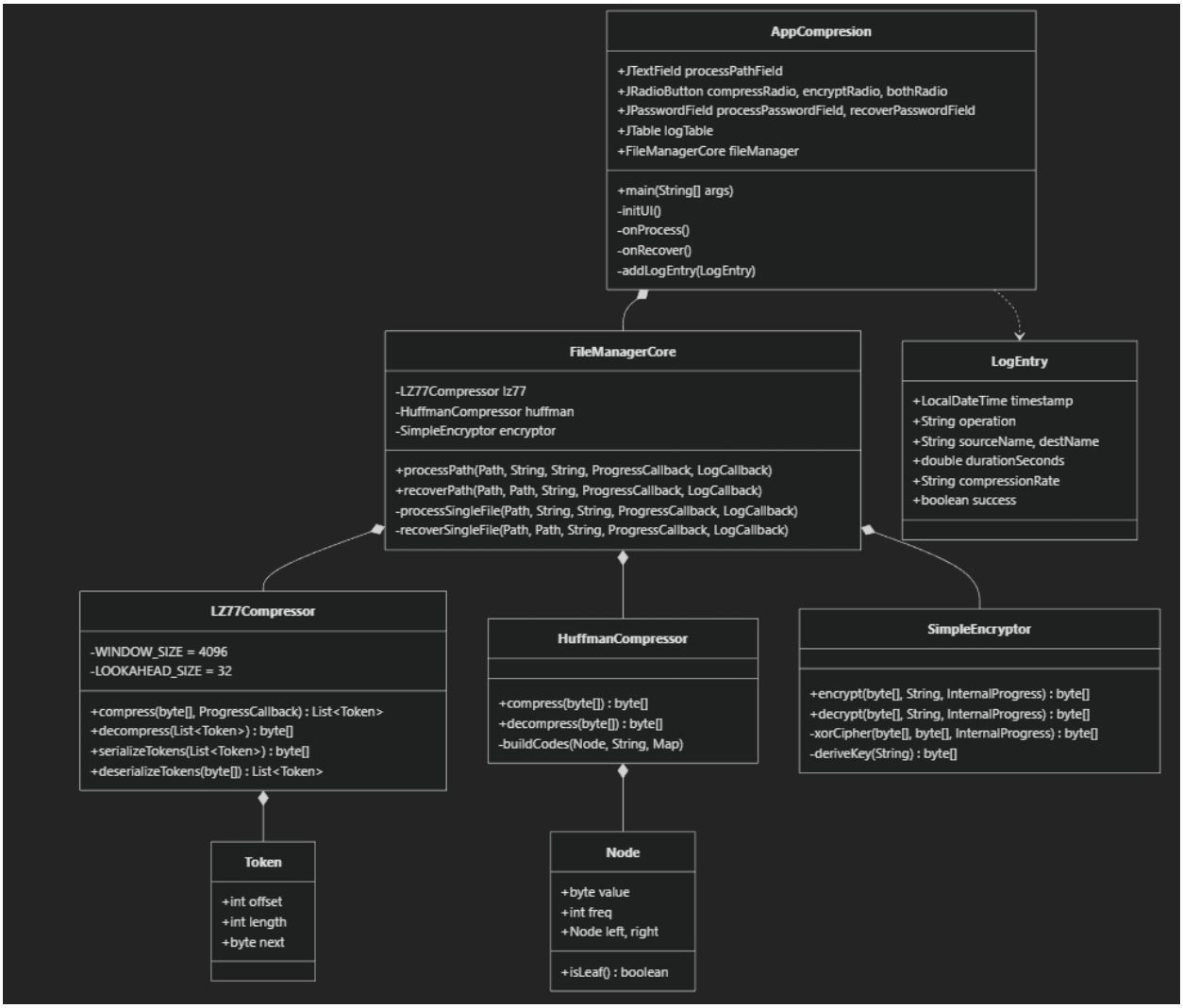
En estos casos los algoritmos LZ77 y el algoritmo de Huffman se encargan de la parte de Compresión y el algoritmo de XOR se encarga de la parte de inversos, se tomó esas decisiones debido a que se evaluaron alternativas vistas en el curso como el algoritmo LZ78 y LZS, pero ninguno funciono de manera correcta mas que nada por la naturaleza que tienen y se utilizo el algoritmo LZ77 no solo porque está libre de Copyright sino también porque es el más rápido para este tipo de tareas recursivas, en este caso se emplea para la reconstrucción de las listas de tokens y es mas rápido, mientras que el algoritmo de Hoffman se encarga de codificar y decodificar en un flujo de bits.

Por ultimo en la parte del algoritmo basado en XOR es un algoritmo de clave derivada, esto se tomo en cuanta debido a que en el proyecto se estableció que tiene que ser posible ponerles clave a los archivos para desencriptarlos entonces se utilizo una clave derivada SHA-256 a partir de la contraseña ingresada por el usuario.

Arquitectura del programa



Clases y variables que se utilizan



En estos casos los algoritmos se expresan de la siguiente forma:

1) LZ77 (Compresión por ventana deslizante)

El algoritmo LZ77 utilizado trabaja sobre una ventana de tamaño fijo y un búfer de anticipación:

Se recorre el archivo de entrada byte por byte.

Para cada posición:

Se busca el match más largo entre los bytes del búfer de anticipación y la ventana de historial.

Si se encuentra una coincidencia de longitud suficiente (por ejemplo, ≥ 3 bytes):

Se genera un token (offset, length, nextByte), donde:

offset = distancia hacia atrás desde la posición actual.

length = número de bytes repetidos.

nextByte = siguiente byte literal después de la secuencia repetida.

Si no hay coincidencia útil:

Se genera un token (0, 0, byteLiteral).

Los tokens se almacenan en una lista de objetos Token.

Este paso reduce la redundancia al representar secuencias repetidas mediante referencias (offset, length).

2) Huffman (Codificación entropía)

Una vez que los tokens LZ77 se serializan a un arreglo de bytes, se aplica Huffman:

Se calculan las frecuencias de cada byte en el flujo de tokens serializado.

Se construye un árbol binario:

Cada hoja representa un símbolo (byte) y su frecuencia.

Se combinan nodos de menor frecuencia hasta formar un árbol completo.

A cada símbolo se le asigna un código binario:

Símbolos más frecuentes → códigos más cortos.

Símbolos raros → códigos más largos.

El archivo comprimido incluye:

Longitud original de los datos.

Cantidad de símbolos distintos.

Tabla (símbolo, frecuencia) para reconstruir el árbol.

Flujo de bits comprimidos.

Esto produce un flujo binario más corto que el original, aprovechando la distribución de frecuencias.

3) Cifrado por XOR con clave derivada

El cifrado se implementa en la clase SimpleEncryptor:

El usuario proporciona una contraseña.

Se deriva una clave de 32 bytes usando SHA-256(password).

Para cifrar/descifrar:

Se recorre el arreglo de bytes de entrada.

Para cada posición i:

Se toma el byte key[i % keyLen].

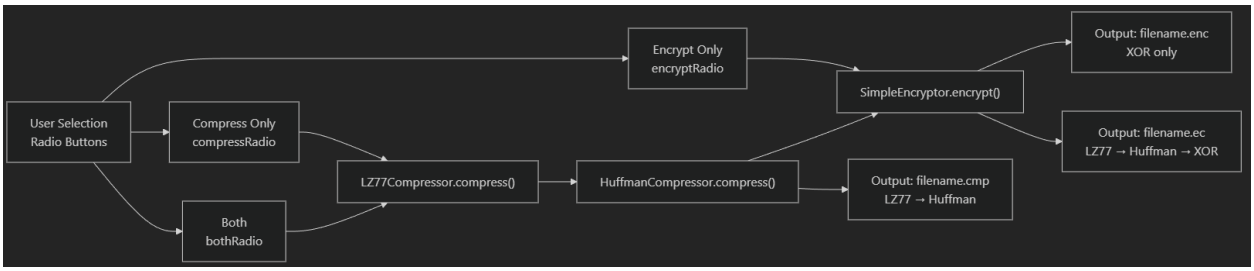
Se combina con la posición (i * 31) para variar un poco el patrón.

Se aplica la siguiente formula

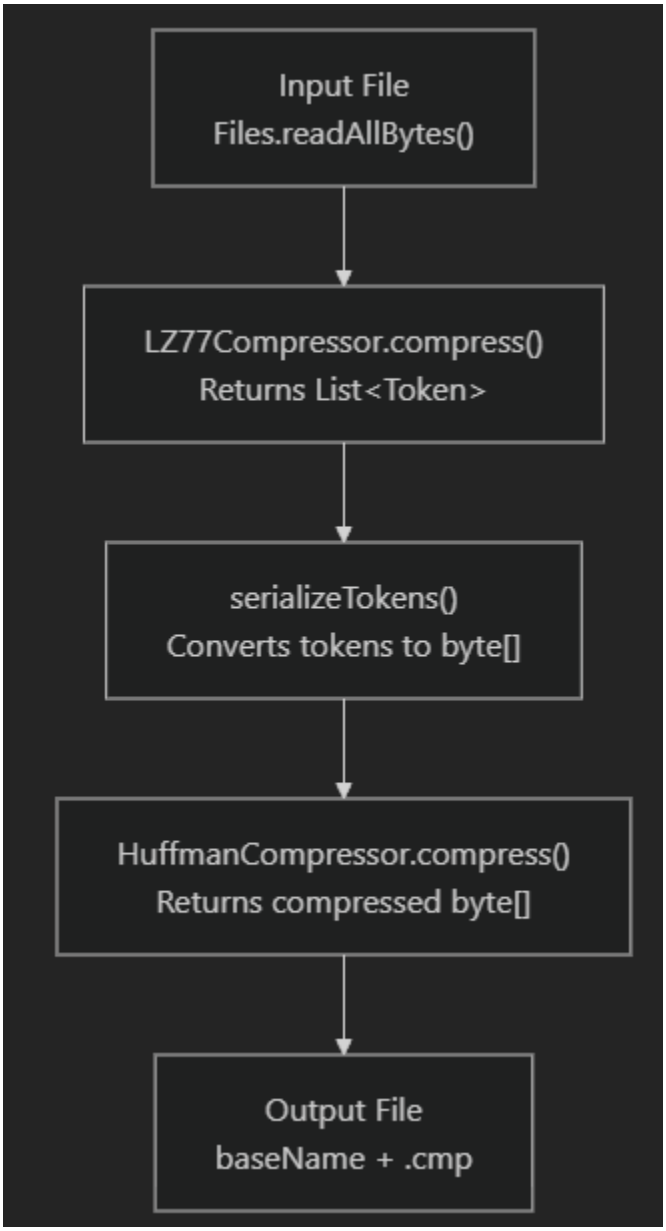
$$out\ (i) = in(i)key(i\ mod\ keyLen)(ix31)$$

La misma operación se usa para cifrar y para descifrar ya que el XOR es simétrico.

Como Funciona el programa



Explicación de la codificación y generación del archivo comprimido



Cuando el usuario selecciona “Solo compresión (.cmp)”:

1. Se lee el archivo original completo en memoria como arreglo de bytes.
2. Se aplica LZ77:

   Se genera una lista de Token (offset, length, nextByte).
3. Se serializan los tokens:

   Primero se escribe el número de tokens (int).

   Luego, por cada token:

   - offset → short
   - length → short
   - next → byte

   El resultado es un flujo de bytes listo para Huffman.
4. Se aplica Huffman sobre ese flujo:

Se genera:

- Longitud original.
- Tabla de frecuencias.
- Flujo de bits comprimido.

5. Se guarda en disco un archivo con extensión .cmp:

   Ubicado en la misma carpeta del archivo original.

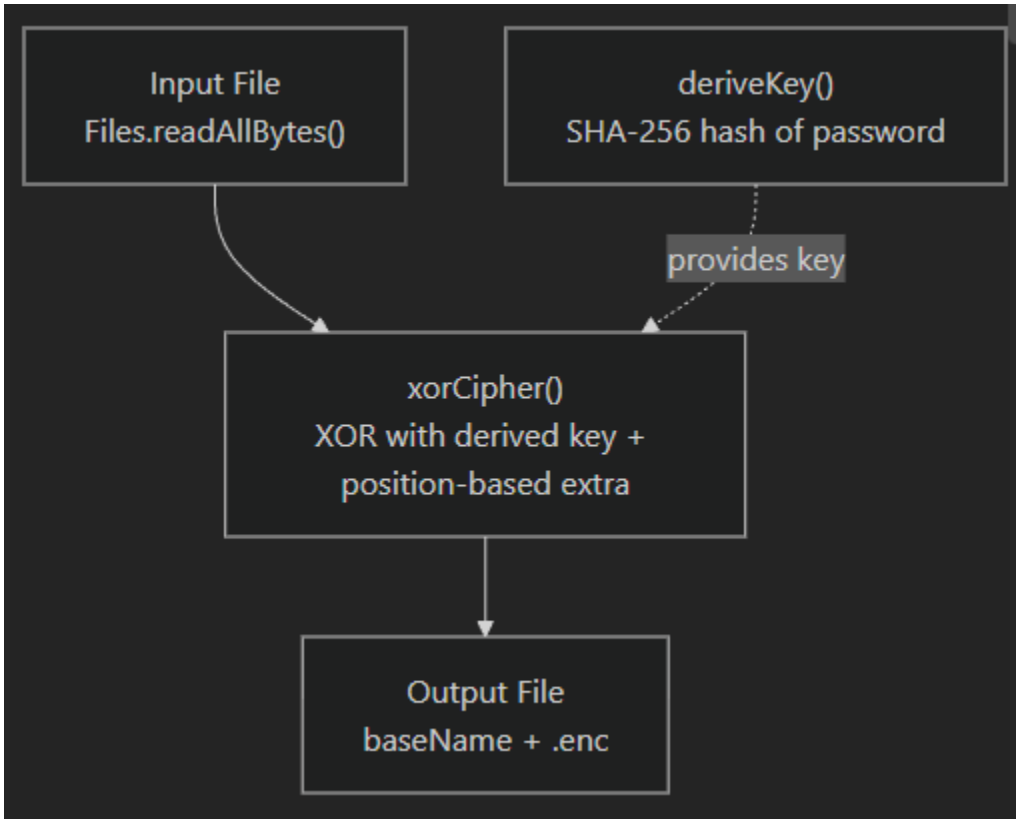
   Nombre base igual al original (sin la última extensión) + .cmp.

Proceso de descompresión del Archivo

Al recuperar un .cmp:

1. Se lee el archivo comprimido.
2. Se reconstruye el árbol de Huffman desde la tabla de frecuencias del encabezado.
3. Se decodifica el flujo de bits a los bytes de tokens serializados.
4. Se desrealiza la lista de tokens (offset, length, nextByte).
5. Se aplica LZ77 en modo inverso, reconstruyendo el flujo original.
6. Se guarda un archivo sin extensión adicional (usando el nombre base original).

Explicación de la encriptación y generación del archivo encriptado



Cuando el usuario selecciona “Solo encriptación (.enc)”:

1. Se lee el archivo original como arreglo de bytes.
2. Se solicita una contraseña al usuario.
3. SimpleEncryptor:

   Deriva una clave de 32 bytes con SHA-256.

   Aplica el cifrado XOR sobre cada byte.
4. Se genera un nuevo archivo con extensión. enc:

   Ejemplo: documento.pdf → documento.enc

Para recuperar:

1. Se lee el. enc.
2. Se solicita la misma contraseña.
3. Se aplica nuevamente el mismo XOR.
4. Se obtiene el archivo original (mismo contenido y tamaño).

Compresión + Encriptación (.ec)

Cuando el usuario selecciona “Compresión + Encriptación (.ec)”:

1. Se leen los bytes del archivo original.
2. Se aplica LZ77 + Huffman (igual que en el caso .cmp).
3. Se toma el resultado comprimido y se aplica cifrado XOR.
4. Se genera un archivo con extensión .ec:
   - Ejemplo: documento.txt → documento.ec

Conclusiones

La aplicación cumple con los requerimientos de:

- Compresión de archivos mediante una combinación de LZ77 + Huffman.
- Encriptación simétrica basada en XOR y contraseña del usuario.
- Recuperación correcta de la información a partir de los archivos comprimidos y/o encriptados.
- Registro detallado (log) de cada operación, incluyendo:
    - Nombre original y nuevo del archivo.
    - Tiempo de procesamiento.
    - Tasa de compresión alcanzada.
    - Estado de la operación (éxito o error).
    - Exportación del log en formato CSV, compatible con Excel para análisis.

El uso combinado de LZ77 y Huffman permite obtener buenas tasas de compresión para archivos con redundancia, al separar el problema en:

- Eliminación de repeticiones (LZ77) y
- Optimización de códigos según frecuencia (Huffman).

El esquema de cifrado XOR con clave derivada por SHA-256 es sencillo, didáctico y suficiente para un proyecto académico, demostrando el uso de:

- Contraseñas,
- Derivación de claves,
- Cifrado y descifrado simétricos.

La interfaz en Java Swing ofrece:

- Separación clara entre procesamiento y recuperación.
- Feedback visual mediante barras de progreso y mensajes de estado.
- Un historial de operaciones útil para:
    - Depuración,
    - Evaluación de rendimiento,
    - Generación de evidencia para el informe del proyecto.

Como posibles trabajos futuros se podrían considerar:

- Implementar algoritmos de compresión alternativos (por ejemplo, solo Huffman sobre datos sin LZ77, o integrar otros esquemas).
- Utilizar algoritmos de cifrado estándar (AES) para una mayor seguridad en entornos reales.
- Añadir validación más estricta de entradas y manejo de errores a nivel de interfaz (por ejemplo, bloqueo de botones mientras hay tareas en curso).