PROJECTES DE PROGRAMACIÓ

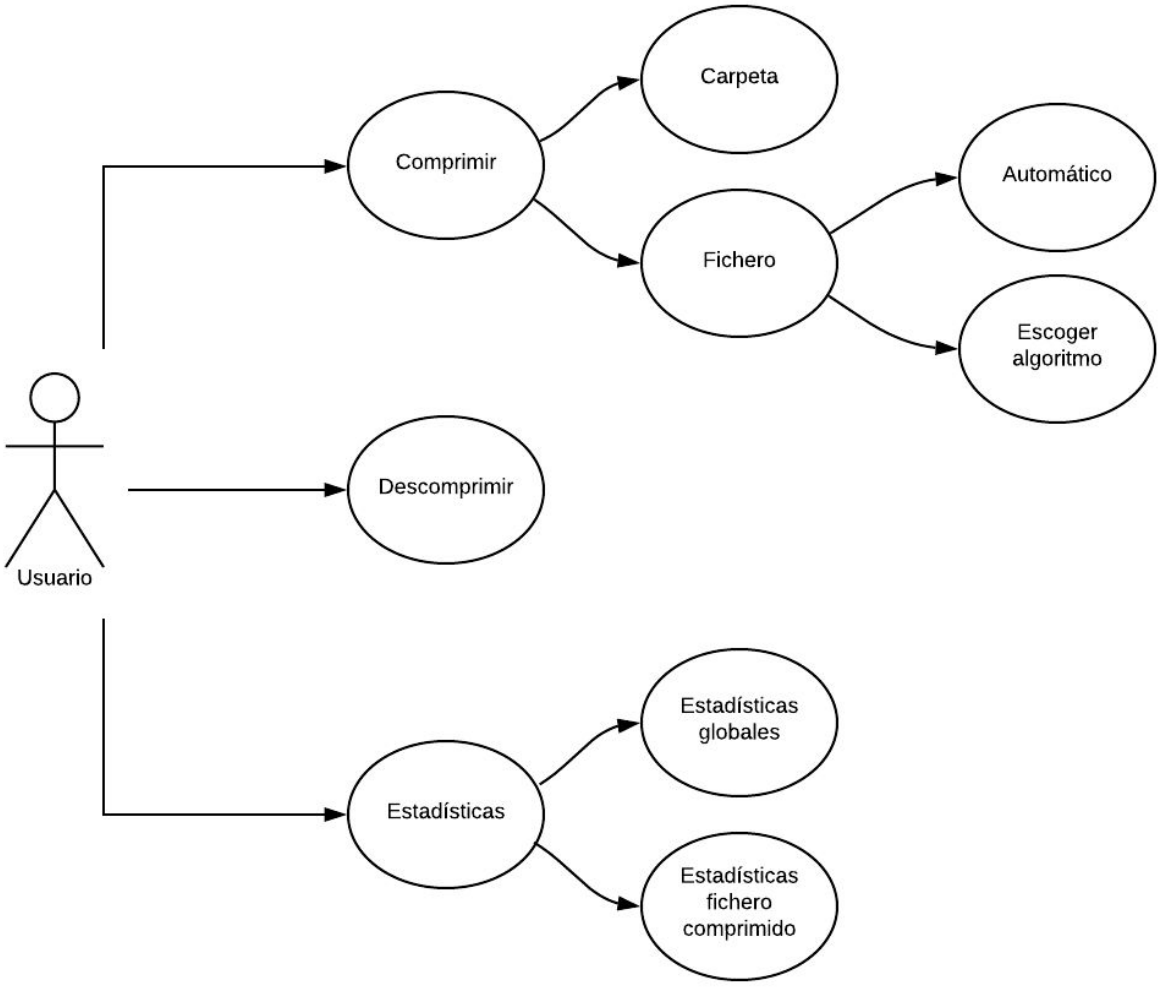
Documentación del proyecto de PROP

Marc Monfort, Felipe Ramis, Daniel Donate y Álvaro Macías

Otoño 2019

Profesor: Borja Vallès

1. Definición de casos de uso



1. Comprimir:

El usuario interactúa con el controlador de presentación. Primero deberá escoger si comprimir un archivo o una carpeta. A través de una interfaz el usuario introduce en el sistema el path de (selecciona) un archivo. El usuario introduce el tipo de selección de algoritmo que elija: manual o automática. Si escoge la selección manual, se le ofrecen las diferentes alternativas según el tipo de fichero a comprimir, de entre las cuales escoge una. Si la selección es automática, el sistema escoge la que considere la opción más adecuada. Una vez seleccionado el algoritmo a aplicar, el controlador de compresión hace las llamadas pertinentes al algoritmo y al controlador de estadísticas para generar el archivo comprimido (con el mismo path y nombre que el original, pero con extensión .prop) y las estadísticas de la compresión.

2. Descomprimir:

Caso similar al de comprimir. El usuario interactúa con el controlador de presentación. A través de una interfaz, introduce en el sistema el path del archivo a descomprimir. El sistema selecciona automáticamente el algoritmo a aplicar, que será siempre el mismo que se usó para comprimir, que obtiene de la metadata del archivo comprimido. El controlador de descompresión efectúa entonces la llamada al algoritmo de descompresión, envía el resultado de la aplicación de dicho algoritmo al controlador de presentación y este se encarga de guardar el nuevo archivo descomprimido en la localización de la que se obtuvo el .prop.

3. Estadísticas:

El usuario interactúa con el controlador de presentación. El usuario escoge qué estadísticas desea consultar: globales (registro de compresión) o de un archivo concreto. Si escoge consultar el registro, el controlador de estadísticas se comunica con el controlador de persistencia para obtener el archivo de registro, que le muestra al usuario. Si escoge la opción de consultar las estadísticas de un archivo concreto, a través de una interfaz selecciona el fichero cuyas estadísticas quiere consultar; el controlador de estadísticas le pedirá la información sobre los parámetros de compresión al controlador de persistencia. El sistema le devuelve al usuario la información requerida.

2. Modelo conceptual de Datos

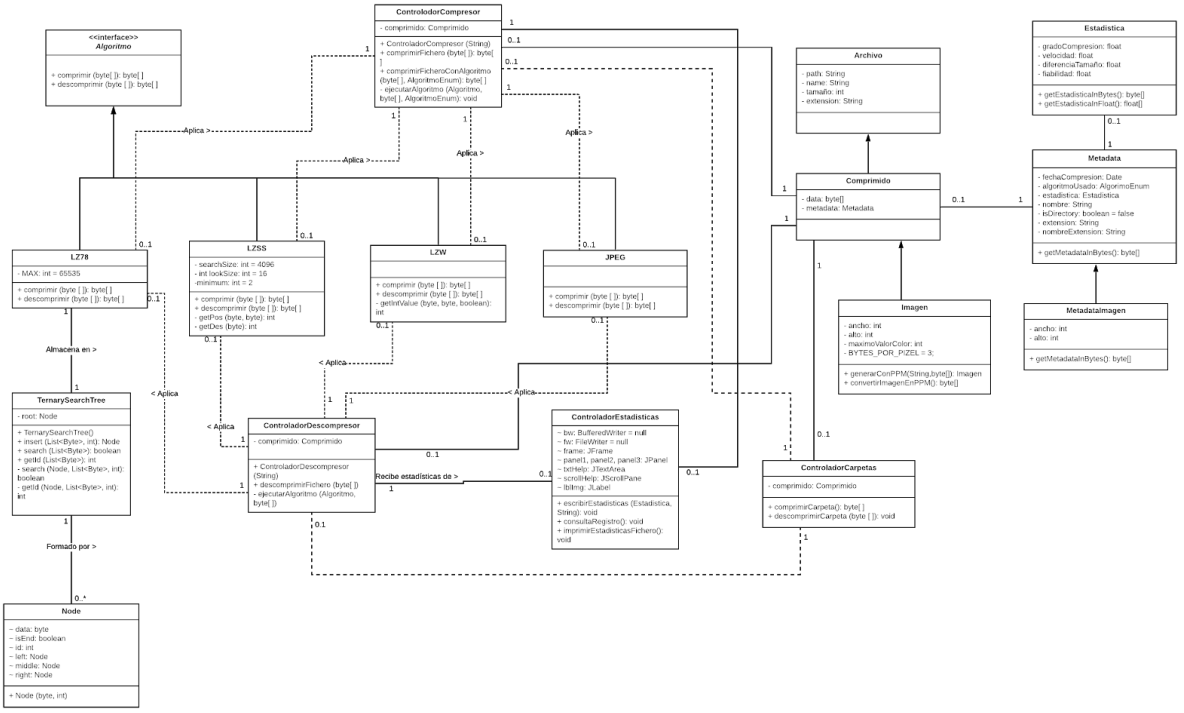


*Imagen adjunta en DiagramaDominio.pdf

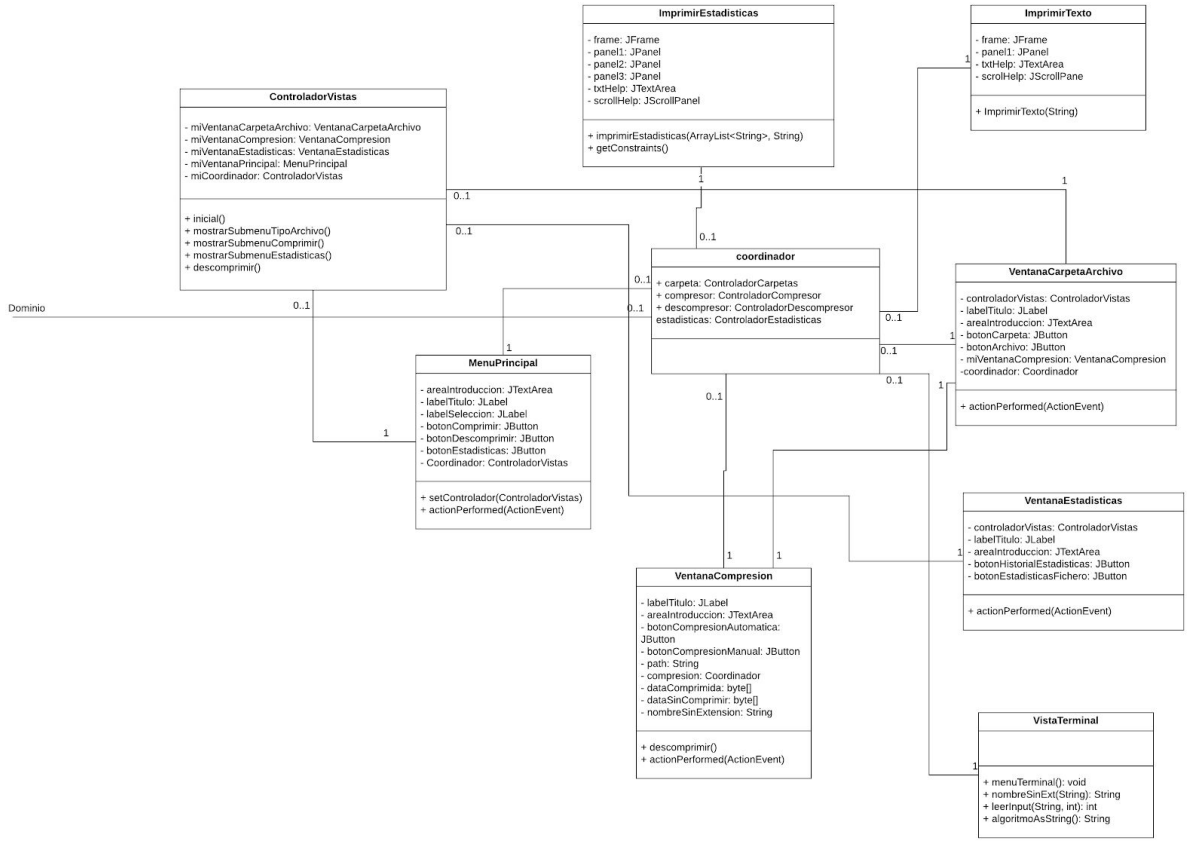


*Imagen adjunta en DiagramaVistas.pdf

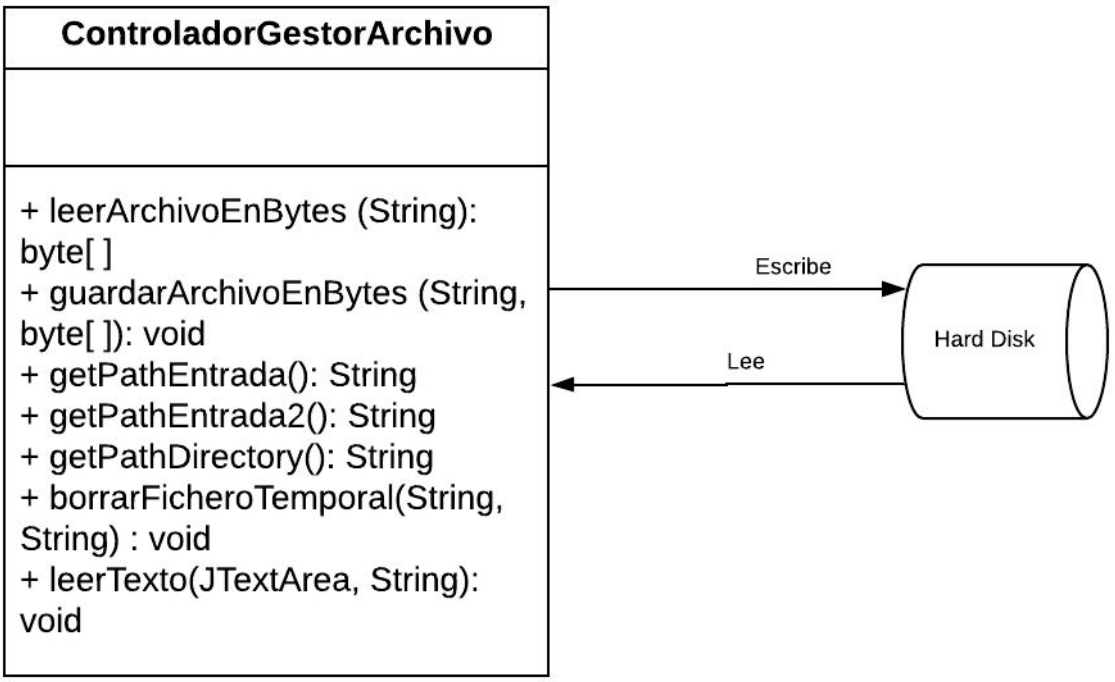


*Imagen adjunta en DiagramaPersistencia.pdf

Dominio

Algoritmo:

Se trata de una interfaz, por lo tanto no implementa ningún método.

AlgoritmoLZ78:

Atributos:

- int MAX = 65535:
  - Índice máximo permitido para los nodos del árbol.

Métodos:

- byte[] comprimir(byte[] entrada):
  - Implementa la compresión con el algoritmo LZ78.
- byte[] descomprimir(byte[] entrada):
  - Implementa la descompresión de un archivo comprimido mediante LZ78.

AlgoritmoLZSS:

Atributos:

- int searchSize = 4096:
  - Tamaño de la ventana de búsqueda.
- int lookSize = 16:
  - Tamaño máximo de las cadenas comparadas.
- int minimum = 2:
  - Tamaño mínimo de las cadenas referenciadas.

Métodos:

- byte[] comprimir(byte[] entrada):
  - Implementa la compresión con el algoritmo LZSS.

- byte[] descomprimir(byte[] entrada):
    - Implementa la descompresión de un archivo comprimido mediante LZSS.

AlgoritmoLZW:

Métodos:

- byte[] comprimir(byte[] entrada):
    - Implementa la compresión con el algoritmo LZW.
- byte[] descomprimir(byte[] entrada):
    - Implementa la descompresión de un archivo comprimido mediante LZW.

AlgoritmoJPEG:

Métodos:

- byte[] comprimir(byte[] entrada):
    - Implementa la compresión con el algoritmo JPEG.
- byte[] descomprimir(byte[] entrada):
    - Implementa la descompresión de una imagen comprimido mediante JPEG.

ControladorCompresor

Atributos:

- Comprimido comprimido:
    - Contiene el resultado de la aplicación de un algoritmo de compresión sobre un archivo.

- Metadata metadata:
    - Contiene la metadata del archivo comprimido.

- Estadistica estadistica:
    - Contiene las estadísticas resultantes de la compresión del archivo.

Metodos:

- ControladorCompresor(String pathARchivo):
  - Constructora de la clase. Crea un objeto del tipo controladorCompresor con acceso al archivo al que dirige el path pasado por parámetro.

- byte[] comprimirFichero(byte[] dataSinComprimir):
  - El parámetro de entrada representa el fichero a comprimir en formato array de bytes, que es el formato que tratan los algoritmos de compresión. Esta función asigna automáticamente, basándose en el tipo y el tamaño del archivo introducido, el algoritmo a aplicar, y efectúa la llamada pertinente a la función de ejecución del algoritmo escogido.

- byte[] comprimirFicheroConAlgoritmo(byte[] dataSinComprimir, AlgoritmoEnum tipoAlgoritmo):
  - El parámetro de entrada representa el fichero a comprimir en formato array de bytes, que es el formato que tratan los algoritmos de compresión. Esta función requiere la elección por parte del usuario del algoritmo con que desea comprimir. Efectúa la llamada pertinente a la función de ejecución del algoritmo escogido.

- ejecturaAlgoritmo(Algoritmo algoritmo, byte[] dataSinComprimir):
  - Esta función se comunica con las clases del dominio para enviar a las clases que implementan los diferentes algoritmos los archivos a comprimir. Recibe por parámetro qué algoritmo debe ejecutarse y el archivo que se desea comprimir.

- String getName():
  - Devuelve el nombre completo del archivo a comprimir.

- String getExtension():
  - Devuelve la extensión del archivo a comprimir.

- String getNombreSinExtension():
  - Devuelve el nombre del archivo a comprimir, sin la extensión.

- Estadística getEstadistica():
  - Devuelve las estadísticas de la compresión.

- AlgoritmoEnum getAlgoritmo():

- Devuelve el código del algoritmo aplicado para la compresión del archivo.

ControladorDescompresor

Atributos:

- Comprimido comprimido:
  - Contiene el archivo a descomprimir.

- Metadata metadata:
  - Contiene la metadata del archivo a descomprimir.

- Estadistica estadistica:
  - Contiene las estadísticas de compresión del archivo a descomprimir.

Métodos:

- ControladorDescompresor(String pathArchivo):
  - Constructora por defecto. Crea un objeto del tipo controladorDescompresor con acceso al archivo indicado en el path introducido por parámetro.

- byte[] descomprimirFichero(byte[] dataComprimida):
  - Esta función recibe por parámetro el archivo a descomprimir. Extrae la información de metadata y estadísticas y prepara el archivo para la ejecución del algoritmo de descompresión, que se determina unívocamente con la información del metadata.

- ejecutarAlgoritmo(Algoritmo algoritmo, byte[] dataSinMetadata):
  - Esta función se comunica con las clases del dominio para enviar a la clase que implementa el algoritmo de descompresión pertinente el archivo a descomprimir. Acepta como parámetros el algoritmo que debe aplicarse y el archivo a descomprimir, ya sin metadata ni estadísticas.

- String getName():
  - Devuelve el nombre original del archivo a descomprimir.

- String getExtension():
  - Devuelve la extensión original del archivo a descomprimir.

- String getNombreSinExtension():
    - Devuelve el a¡nombre original del archivo a descomprimir, sin extensión.

ControladorGestorArchivo

Métodos:

- byte[] leerArchivoEnBytes(String path):
    - Convierte el archivo localizado en el path pasado por parámetro a formato array de bytes para que pueda ser procesado por los algoritmos de compresión.

- guardarArchivoEnBytes(String path, byte[] data):
    - Guarda una entrada en formato array de bytes en un archivo en el path indicado por parámetro.

- String getPathEntrada():
    - Permite al usuario seleccionar un archivo. Obtiene el path de dicho archivo. Esta acción puede cancelarse sin interrumpir la ejecución del programa.

- String getPathEntrada2():
    - Permite al usuario seleccionar un archivo, con la restricción de que solo se pueden seleccionar archivos .prop para ser descomprimidos. Obtiene el path de dicho archivo. Esta acción puede cancelarse sin interrumpir la ejecución del programa.

- void logError(String mensajeError):
    - En caso de error imprime por pantalla un mensaje informativo.

- String getCurrentDirectory():
    - Devuelve el directorio en que se encuentra el usuario.

ControladorEstadisticas

Atributos:

- BufferWriter bw = null:
- FileWriter fw = null:

Métodos:

- void escribirEstadisticas(Estadistica estadistica, String algoritmoUsado):
  - Escribe las estadísticas de compresión de un archivo. Si es la primera vez que se generan estadísticas, crea un archivo con la información. Si no, las estadísticas se añaden al archivo ya existente.

- void consultarRegistro():
  - Abre una vista para consultar el registro de estadísticas del sistema, en caso de que dicho registro exista.

- void imprimirEstadisticasFichero():
  - Muestra las estadísticas de un fichero concreto seleccionado por el usuario a través de una vista.

ControladorDominio:

Atributos:

- ControladorCarpetas carpeta
- ControladorCompresor compresor
- ControladorDescompresor descompresor
- ControladorEstadisticas estadisticas

Métodos:

- void setControladorCompresor():
  - Setter del Controlador Compresor.

- void setControladorDescompresor():
  - Setter del Controlador Descompresor.

- void setControladorEstadisticas():
  - Setter del Controlador Estadísticas.

- void setControladorCarpetas(String path):
    - Setter del Controlador de Carpetas.

- void leerTexto(JTextArea txtHelp, String path_final):
    - Enlace entre las vistas y una funcionalidad del Gestor de Archivos.

- void getPathDirectory():
    - Enlace entre las vistas y una funcionalidad del Gestor de Archivos.

- void guardarArchivoEnBytes(String path_final, byte[] dataComprimida):
    - Enlace entre las vistas y otra funcionalidad del Gestor de Archivos.

- String getPathEntrada():
    - Enlace entre las vistas y otra funcionalidad del Gestor de Archivos.

- String getPathEntrada2():
    - Enlace entre las vistas y otra funcionalidad del Gestor de Archivos.

- byte[ ] leerArchivoEnBytes(String path):
    - Enlace entre las vistas y otra funcionalidad del Gestor de Archivos.

- byte[ ] descomprimirFichero(byte[] dataComprimida):
    - Enlace entre las vistas y una funcionalidad del Gestor Descompresor.

- byte[ ] comprimirCarpeta():
    - Enlace entre las vistas y otra funcionalidad del Gestor de Carpetas.

- void descomprimirCarpeta(byte[] dataComprimida):
    - Enlace entre las vistas y otra funcionalidad del Gestor de Carpetas.

- String getNameCarpeta():
    - Enlace entre las vistas y otra funcionalidad del Gestor de Carpetas.

- byte[ ] comprimirFichero(byte[] dataSinComprimir):
    - Enlace entre las vistas y una funcionalidad del Controlador Compresor.

- byte[ ] comprimirFicheroConAlgoritmo(byte[] dataSinComprimir, int opcion):
    - Enlace entre las vistas y otra funcionalidad del Controlador Compresor.

- void consultarRegistro():
    - Enlace entre las vistas y una funcionalidad del Controlador de Estadísticas.

- void imprimirEstadisticasFichero():
    - Enlace entre las vistas y otra funcionalidad del Controlador de Estadísticas.

- String getNameDescomprimido():
    - Obtiene el nombre de un fichero descomprimido.

- String getNameDescomprimidoSinExtension():
    - Obtiene el nombre de un fichero descomprimido sin extrensión.

- String getExtension():
    - Obtiene la extensión de un fichero descomprimido.

- String getAlgoritmoUsadoCompresor():
    - Obtiene el algoritmo empledo en la compresión de un archivo.

- String getAlgoritmoUsadoDescompresor():
    - Obtiene el algoritmo empledo en la descompresión de un archivo.

- String getVelocidad():
    - Obtiene la velocidad con la que se ha descomprimido un archivo.

- float[ ] getEstadisticasComprimido():
    - Obtiene las estadísticas de un fichero comprimido.

Vistas:

ControladorVistas:

Atributos:

- ventanaCarpetaArchivo miVentanaCarpetaArchivo
- ventanaCompresion miVentanaCompresion
- ventanaEstadisticas miVentanaEstadisticas
- menuPrincipal miVentanaPrinicpal

Métodos:

- void setMiVentanaPrincipal():
  - Enlace con la ventana del menú principal.
- void setMiVentanaCarpetaArchivo():
  - Enlace con la ventana de selección de tipo de elemento a comprimir.
- void setMiVentanaCompresion():
  - Enlace con la ventana de selección de tipo de compresión.
- void setMiVentanaEstadisticas():
  - Enlace con la ventana del menú de estadísticas.
- void mostrarSubmenuTipoArchivo():
  - Mostrar la vista SubmenuTipoArchivo.
- void mostrarSubmenuComprimir():
  - Mostrar la vista VentanaCompresion.
- void mostraSubmenuEstadisticas():
  - Mostrar la vista VentanaEstadisticas.
- void descomprimir():
  - Llama a la funcionalidad de descompresión del sistema.

3. Descripción de las EDA

TernarySearchTree:

Árbol de búsqueda ternario. Consiste en una estructura arbórea simplemente encadenada. Cada nodo contiene un caracter y un identificador, de manera que una rama forma una palabra. Así mismo, contienen también un valor booleano que indica si el nodo es o no una hoja. A cada nodo se le permiten un máximo de tres hijos, y se estructuran de la siguiente forma:

- Si al insertar un nuevo nodo su contenido es lexicográficamente menor que el contenido del padre, se inserta como hijo izquierdo.
- Si el contenido del nuevo nodo es lexicográficamente mayor que el del padre, el nuevo nodo se inserta como hijo derecho.
- Si el contenido pertenece al prefijo, se inserta como hijo central.

Esta estructura implementa también métodos de búsqueda para determinar si una palabra existe en el árbol, y para obtener el identificador de una palabra.

AlgoritmoLZ78:

El algoritmos de compresión LZ78 consigue comprimir los textos buscando cadenas de caracteres repetidas y sustituyéndolas por un puntero al prefijo repetido. Los datos a comprimir o descomprimir son tratados a nivel de bytes, así evitamos los problemas relacionados con caracteres especiales (ç, ñ, etc) y nos permite comprimir cualquier tipo de archivo, no limitándonos a archivos de texto (aunque posiblemente con un ratio de compresión más bajo).

Para implementar la función de compresión he utilizado el TernarySearchTree. Esta estructura de datos es ideal para poder generar el diccionario al comprimir. Mejoramos tanto la velocidad como el espacio requerido.
Recorremos el texto original carácter a carácter, y en cada iteración comprobamos si ese carácter ya aparece anteriormente. Si así es, obtenemos su índice de nuestro diccionario (TernarySearchTree). En la siguiente iteración buscamos si ya hemos visitado un cadena de caracteres con el nuevo carácter y el anterior. Seguimos hasta que no encontramos más repeticiones, y nos guardamos el último índice encontrado y el último carácter.
La compresión genera una cadena de pares: un puntero (2 bytes) y un caracter (1 byte). El puntero es un índice que representa el prefijo del carácter de la palabra representada.

La función de descompresión utiliza un hashMap. En este caso el TernarySearchTree no permite mejorar ni la velocidad ni el espacio, ya que ahora necesitamos generar un mapeo de enteros a cadenas de caracteres.
Únicamente lee los datos comprimidos y genera la salida original (sin pérdida de datos) siguiendo la misma estructura de un puntero y un carácter.

AlgoritmoLZSS:

Implementación del algoritmo de compresión/descompresión LZSS. La implementación sigue el patrón estándar de este algoritmo. La única diferencia notable es el tratamiento de la *sliding window*. Normalmente, esta se implementa utilizando dos vectores: uno para el *search buffer* (vector que contiene los caracteres con entre los que se espera encontrar coincidencias) y otro para el *look ahead* (vector que contiene la cadena de caracteres que esperamos comprimir). Estos vectores se deben ir llenando con nueva información a medida que iteramos sobre el fichero a comprimir, como si fuera una ventana corrediza sobre el archivo. Una mejora común es que el *search buffer* sea un vector circular, de manera que siempre se sustituyen las cadenas más antiguas. Sin embargo, al procesar la entrada (archivo a comprimir) como un vector de bytes (byte[]), en esta implementación no ha sido necesaria ninguna estructura auxiliar para guardar la ventana. Estas se han simulado iterando sobre el propio byte[] de la entrada. Esto supone una mejora de eficiencia tanto espacial (nos evitamos dos estructuras auxiliares) y temporal (no escribimos en, ni eliminamos de, un vector como sugiere la implementación, iteramos sobre la entrada y la escritura del vector se sustituye por el cambio de valor del iterador).

AlgoritmoLZW:

El algoritmo de compresión LZW fue pensado como una mejora de su predecesor, el LZ78, explicado anteriormente. Una diferencia que existe entre ellos se da al principio de la ejecución, pues el LZW inicializa su diccionario con una precarga de 256 entradas (una para cada carácter, siguiendo el diccionario ASCII). A este diccionario se le van añadiendo sucesivos códigos numéricos por cada nuevo par de caracteres sucesivos que se leen.
Estas entradas pueden representar secuencias de caracteres simples o secuencias de códigos, de tal manera que un código puede representar: o bien dos caracteres, o bien secuencias de códigos previamente cargados. De este modo, un código puede “simbolizar” desde uno a un número indeterminado de caracteres (nótese que, en realidad, el realizar la precarga del diccionario implica la inexistencia de caracteres simples, pues éstos mismos son códigos

dentro del diccionario). Toda esta estructura se representa mediante un diccionario de clave *String* y valor *Integer*. En particular se ha escogido el *hashMap* (pues es cómodo, tiene todas las funcionalidades que requiere la creación de un diccionario).
Una potencial mejora (en términos espaciales, sobretodo) sería la migración de esta implementación a una versión con TST, como ha hecho mi compañero en el LZ78.

AlgoritmoJPEG:

El algoritmo de compresión JPEG se basa en una serie de aplicaciones sobre los píxeles para que luego sean más fácilmente compresible. En general la implementación sigue todos los pasos del JPEG estándar que incluye:

1. Transformación del espacio de color de RGB a YCbCr
2. Downsampling: Se ha seguido el patrón 4:4:4 con lo que no se aplica
3. Separación en bloques: Se separan los espacios de color en 3 vectores y luego cada en bloques de 8x8 píxeles.
4. Transformación discreta del coseno: Se aplica para llevar la representación al dominio de frecuencias
5. Cuantización: Se aplica por defecto la matriz de 50% de calidad de compresión
6. Entropy coding: Se pasa la matriz a un vector recorrido en zigzag y se comprime a base de eliminar los ceros de la diagonal de abajo poniendo un centinela para avisar cuando se pasa entre bloques (para la descompresión).

La estructura de datos sufre varios cambios en la compresión:

1. Vector entrada de bytes se separa en 3 vectores de enteros (ycbcr).
2. Cada vector de entero se transforma en MCU(unidad mínima de JPEG), bloques de 8x8 valores. Se utilizan arrays de 3 dimensiones que funcionan como un vector de matrices bidimensionales, por tanto, están ordenados de izquierda a derecha y de arriba a abajo teniendo en cuenta el ancho y alto de la imagen.
3. Se crean nuevos bloques 8x8 para DCT y Q, pero son doubles en vez de ints ya que se generan al pasar esos algoritmos.
4. Los bloques finales se pasan a un arraylist (lista dinámica) de enteros tras pasar el zigzag y el encoding (ya que pueden haber de 1 a 64 enteros).
5. Finalmente se transforma en un array de bytes y se escribe en memoria.

Para la descompresión se aplican todos los pasos a la inversa.

Vale la pena mencionar que en el encoding de Huffman se debería aplicar una transformación a símbolos para eliminar los ceros intermedios, pero al final se ha dejado la compresión sólo eliminando los ceros a la derecha.

4. Clases implementadas

Daniel Donate:
- AlgoritmoLZW. java
- Main.java
- ControladorEstadisticas.java
- ControladorCompresor.java
- ControladorVistas.java
- imprimirEstadisticas.java
- imprimirTexto.java
- menuPrincipal.java
- ventanaCarpetaArchivo.java
- ventanaCompresion.java
- ventanaEstadisticas.java
- ControladorDominio.java

Álvaro Macías:
- AlgoritmoLZSS.java
- Archivo.java
- Comprimido.java
- Estadistica.java
- Metadata.java
- Algoritmo.java
- Main.java
- VistaTerminal.java
- ControladorEstadistica.java
- ControladorVistas.java
- ControladorGestorArchivo

Marc Monfort:
- AlgoritmoLZ78.java
- TernarySearchTree.java
- DriverTernarySearchTree.java
- Main.java
- Comprimido.java
- Metadata.java
- Estadistica.java
- ControladorCompresor.java
- ControladorDescompresor.java
- ControladorCarpeta.java

Felipe Ramis:

- AlgoritmoJPEG.java
- Main.java
- Imagen.java
- Algoritmo.java
- ControladorGestorArchivo.java
- ControladorCompresor.java
- Comprimido.java
- ByteUtils.java
- DCT.java
- JPEGUtils.java
- Huffman.java
- MetadataImagen
- VentanaCompresion

5. Librerías externas utilizadas

Para implementar el navegador de archivos que permite seleccionar los archivos a comprimir, así como para otras vistas, se ha usado la librería **javax.swing**.

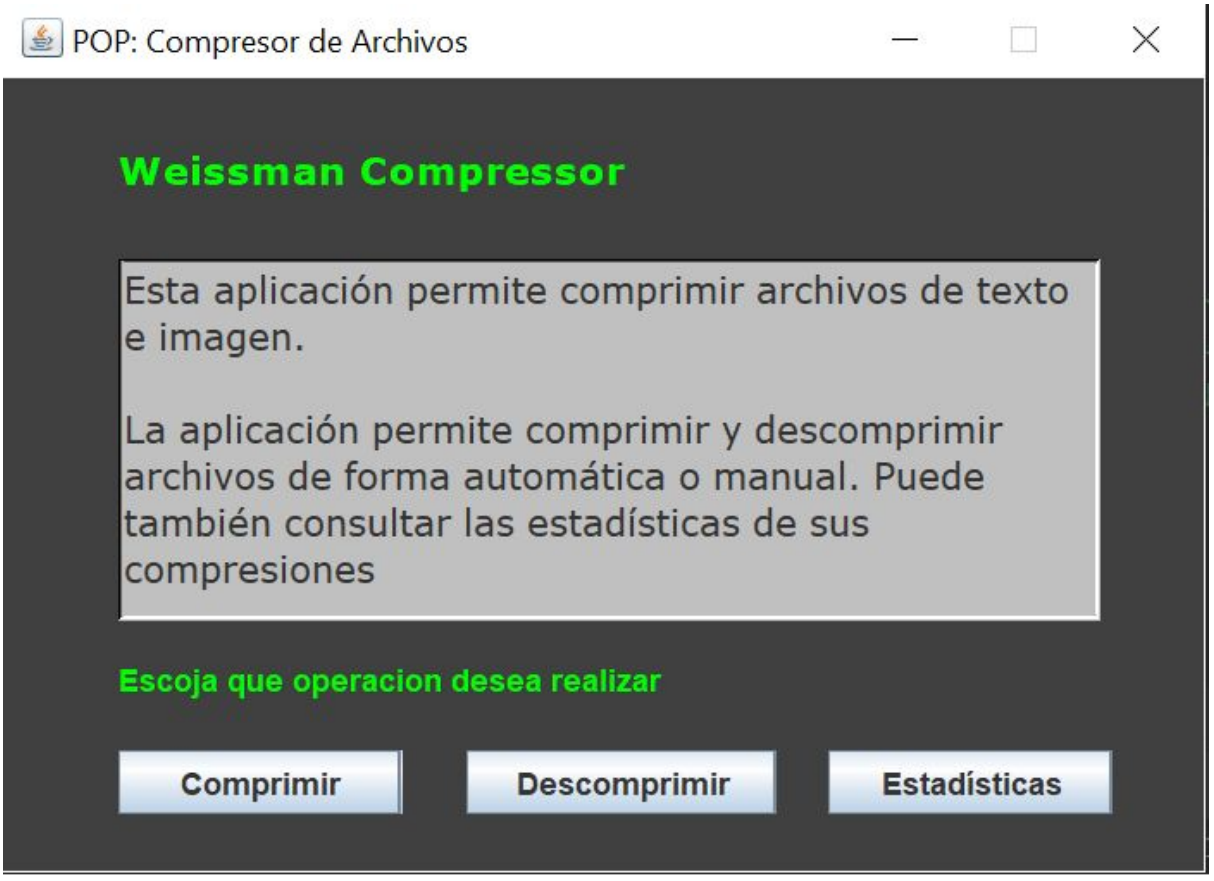
6. Manual de usuario

Para compilar, ejecutamos desde la carpeta subgrup9-2 el comando “make”.

6.1: Versión con GUI

1- Acceder a la aplicación:

Ejecutamos desde terminal, desde la carpeta subgrup9-2, el comando “make runGUI” o bien “make runjar”. Se nos muestra una breve introducción con las funcionalidades del programa, que son las que se muestran a continuación:

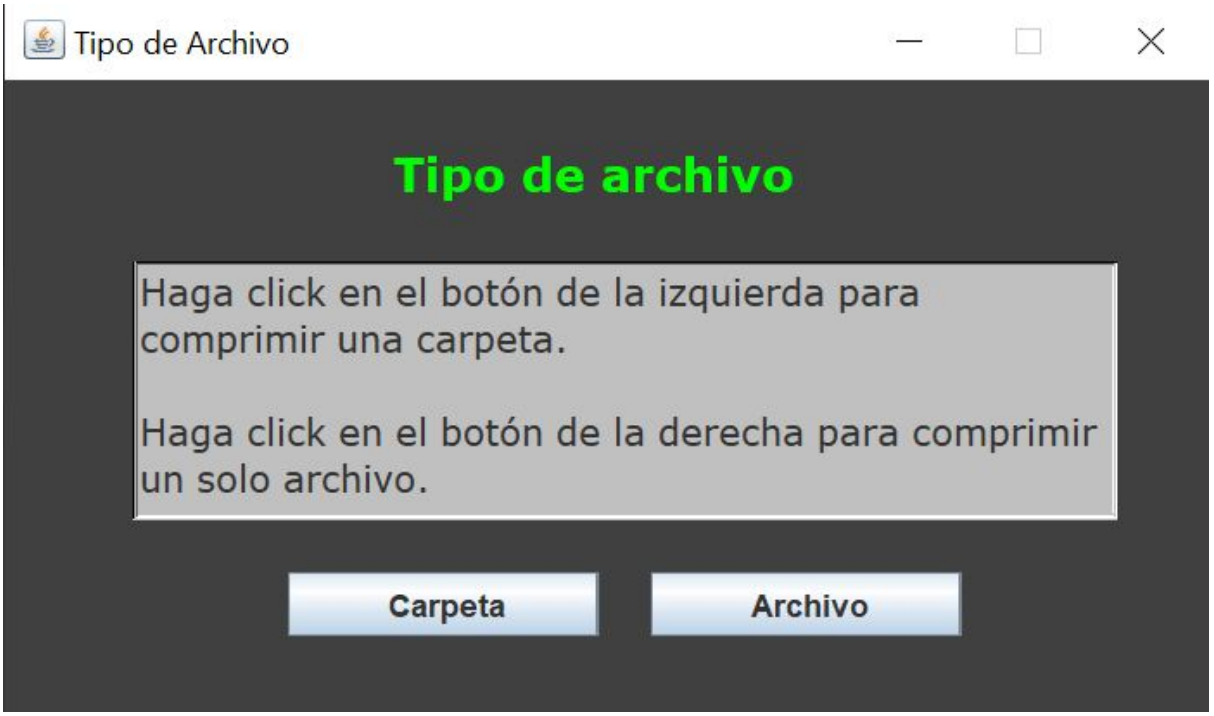


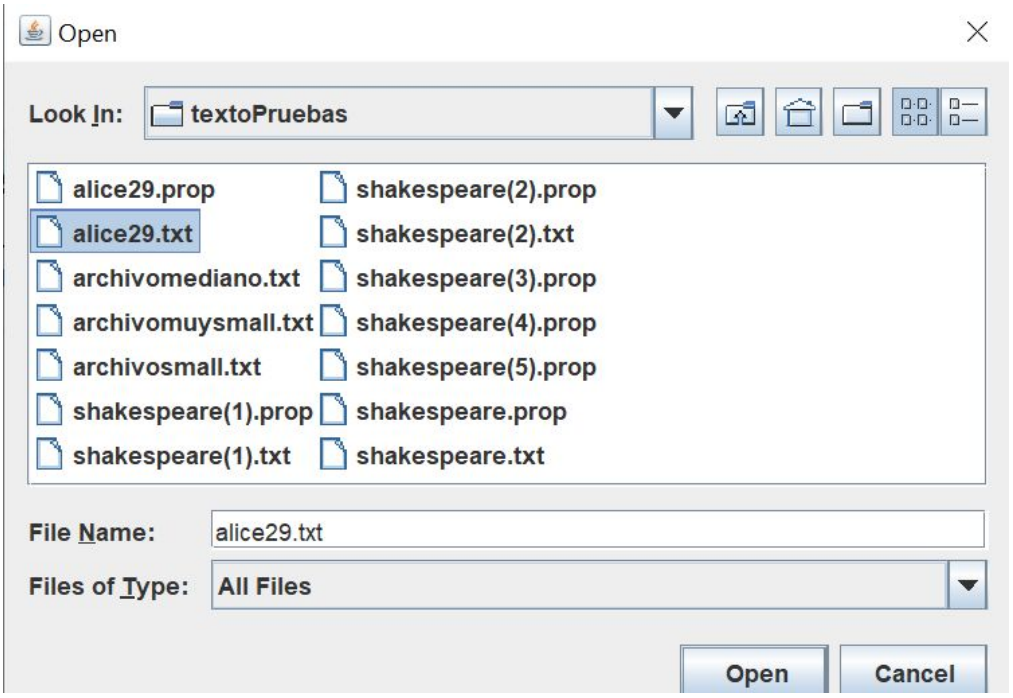
Luego, se nos pide que escojamos una de las 3 opciones seleccionables con los botones.

2- Comprimir:

Si escogemos la opción “Comprimir”, se nos abra una ventana *pop-up* donde se nos pide que escojamos entre comprimir un archivo o comprimir una carpeta. Cualquiera de las dos opciones produce el desplegamiento de un sistema de ficheros que nos permite navegar (desde la carpeta donde estamos ejecutando la aplicación), al archivo o carpeta, respectivamente, que deseamos comprimir.

En la siguiente imagen se muestra la ventana mencionada. Para esta demostración se va a proceder a realizar la compresión de un archivo de texto individual.

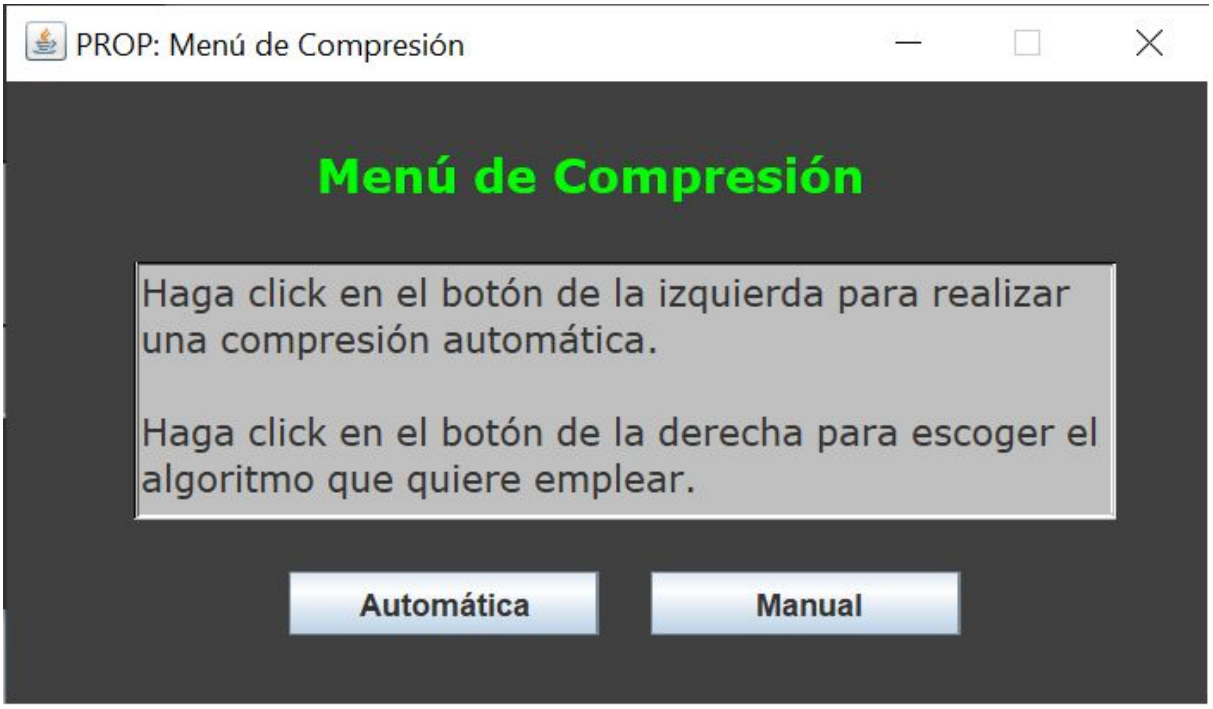




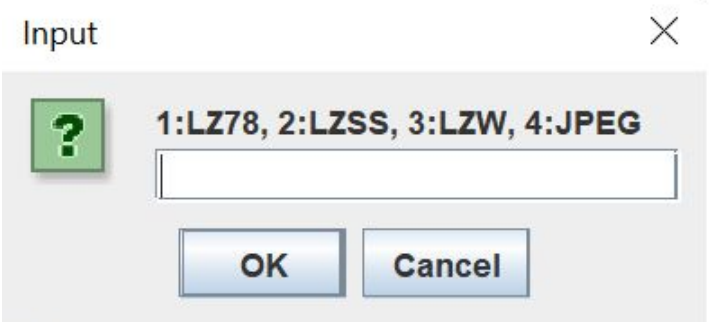
En la anterior imagen, estamos seleccionando el documento “alice29.txt”, que se encuentra en la subcarpeta “textoPruebas”.

2.1- Selección del tipo de compresión:

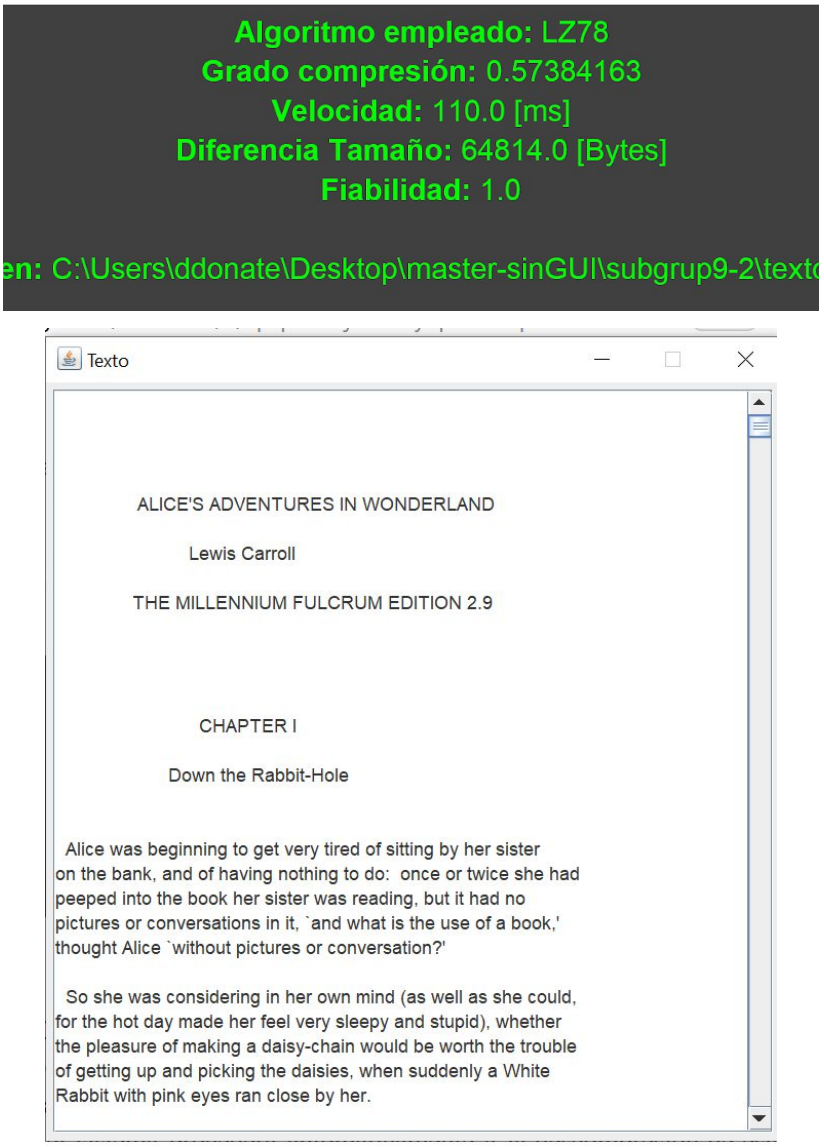
Una vez escogido el fichero que vamos a comprimir, el sistema nos pide que escojamos entre realizar una compresión “Manual” o “Automática” (Nota: el modo manual solo está disponible para ficheros individuales, no para carpetas).



La primera opción hace que sea el programa quién se encargue de seleccionar el algoritmo apropiado para comprimir el contenido que el usuario ha insertado previamente.
En cambio, si el usuario escoge la opción "Manual", se le pedirá que elija cuál de los siguientes 4 algoritmos es el que quiere utilizar.



Una vez escojamos un algoritmo, el programa procederá a la compresión del archivo seleccionado, generando un nuevo fichero con la extensión ".prop", que se guardará en la misma carpeta donde estaba el archivo original.
Finalmente, el sistema nos muestra las estadísticas asociadas al proceso de compresión (concretamente, el grado de compresión, la velocidad, la diferencia de tamaño y el grado de fiabilidad de la compresión).
La siguiente captura muestra un ejemplo para el archivo anteriormente seleccionado ("alice29.txt") utilizando (vía decisión manual) el algoritmo LZ78 junto con una previsualización del archivo comprimido (esta funcionalidad solo está disponible para archivos individuales, no para carpetas).



3- Descomprimir:

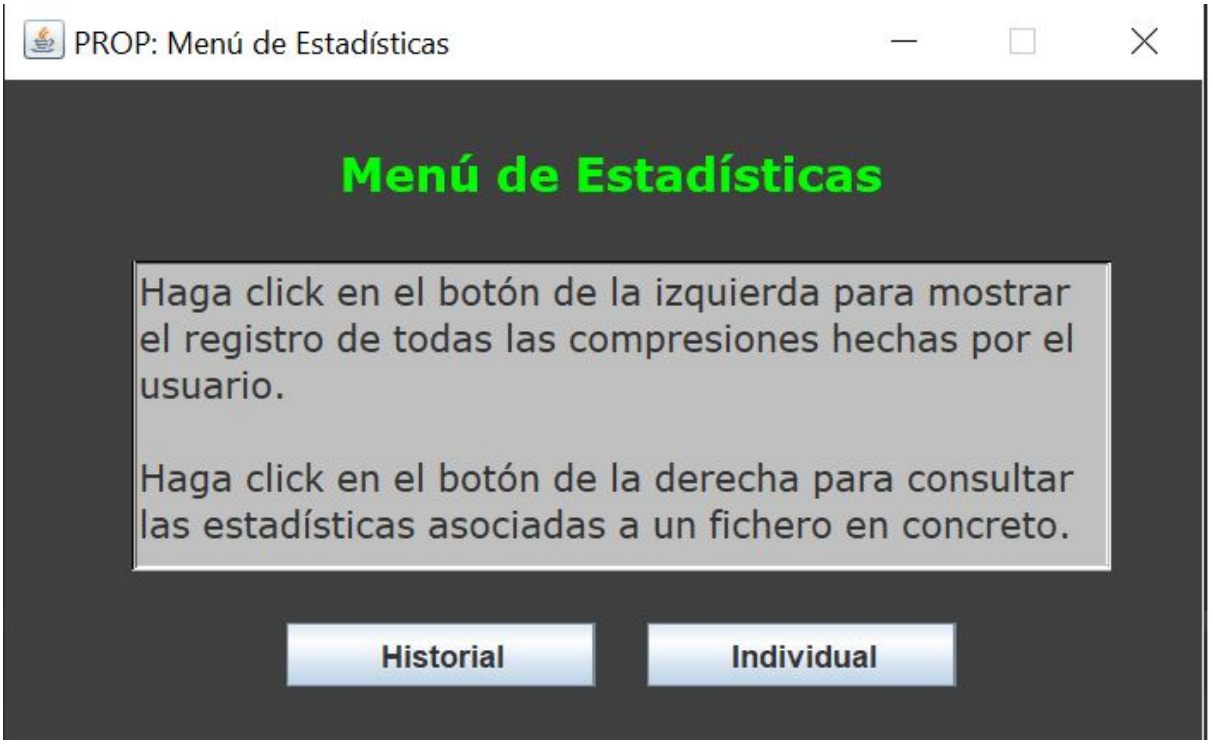
La opción descomprimir nos desplegará, nuevamente, un file chooser que imprimirá, de forma predeterminada, aquellos ficheros que tengan extensión ".prop". Cuando seleccionemos uno, el sistema procederá automáticamente a la descompresión de éste para, posteriormente, guardar en ese mismo directorio el archivo descomprimido (con su extensión

original) y, igual que en la compresión (si el contenido es un archivo individual), mostrar una previsualización del contenido.

Para imágenes, en la visualización se abrirá el visor de imágenes por defecto del sistema operativo.

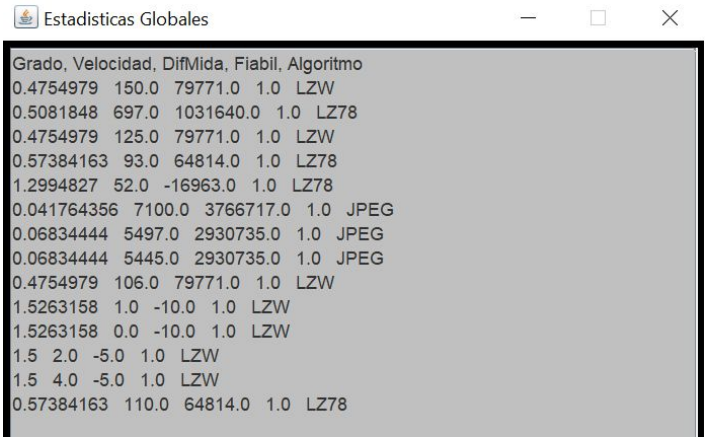
4- Estadísticas:

Si seleccionamos la opción “Estadísticas”, el sistema nos preguntará qué tipo de consulta queremos realizar. Tenemos las siguientes dos opciones:



4.1- Historial de estadísticas:

La elección de la primera opción nos mostrará un registro de todas las compresiones que hemos realizado (la información que se nos muestra es la misma que se nos imprime después de realizar una compresión). En esta imagen se muestra qué aspecto tiene este historial.



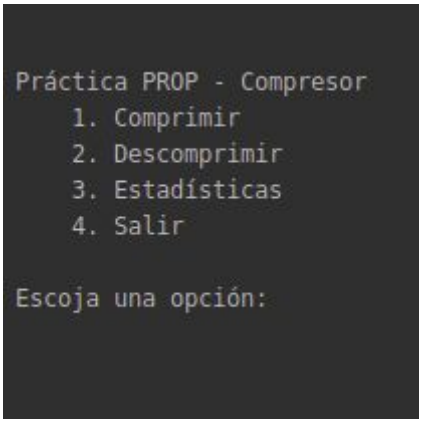
4.2- Estadísticas de un fichero comprimido:

La segunda opción nos despliega nuevamente el selector de ficheros para que introduzcamos un archivo con extensión ".prop" y visualizemos las estadísticas asociadas a ese fichero en particular.

6.2: Versión con terminal

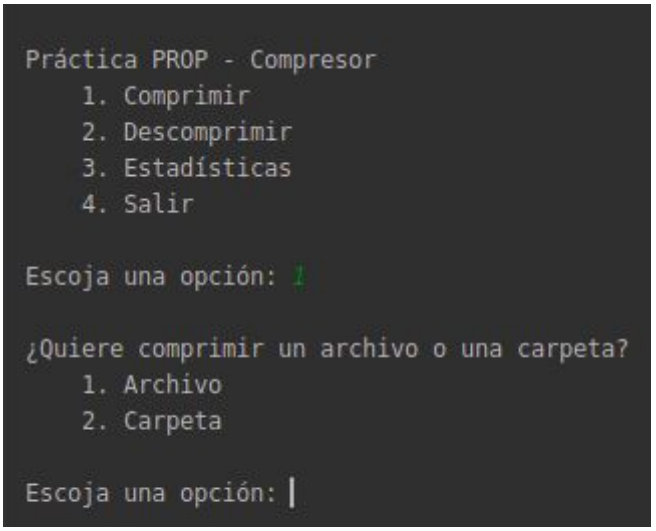
1- Acceder a la aplicación:

Ejecutamos desde la carpeta subgrup9-2 el comando "make runConsole". Se nos muestra un menú con cuatro opciones disponibles:



2- Comprimir:

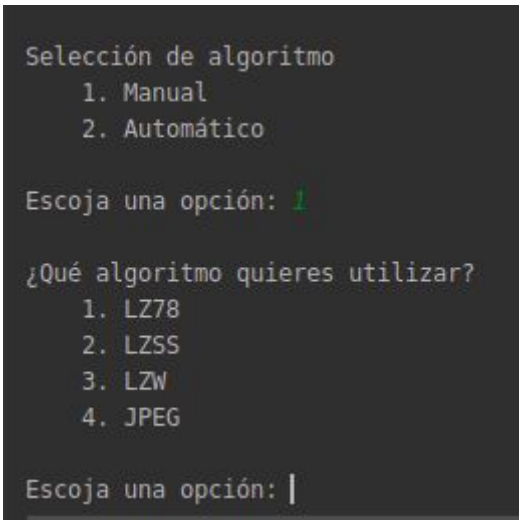
Si escogemos la opción 1 (comprimir), se nos muestra ahora un nuevo menú, preguntando si deseamos comprimir un fichero o una carpeta. En cualquiera de los dos casos, a continuación se nos pide el *path* del archivo o carpeta a comprimir.
En la siguiente imagen se muestra dicho menú:



2.1- Selección del tipo de compresión:

Si hemos escogido comprimir un fichero, una vez introducido el *path* se nos pregunta si queremos comprimir mediante un algoritmo concreto, o si dejamos la decisión al programa.

Si escogemos la opción “Manual”, se nos pedirá que elijamos cuál de los 4 algoritmos disponibles queremos utilizar.



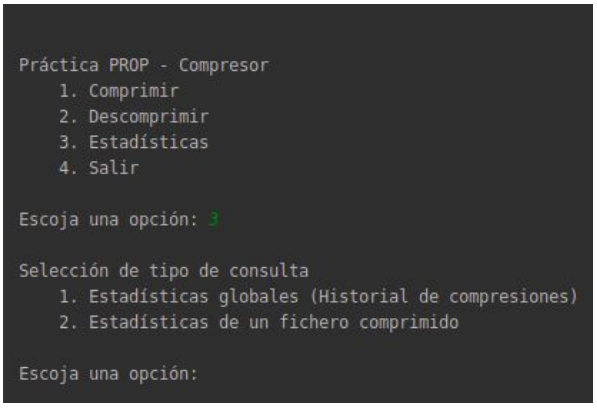
En cualquier caso, se procederá a la compresión. El resultado será un archivo con extensión .prop que se guardará en la misma carpeta que contiene el fichero o carpeta que acabemos de comprimir.

3- Descomprimir

La opción descomprimir funciona esencialmente igual que la de comprimir. Salvo porque ahora no se nos preguntará si deseamos tratar un archivo o carpeta, ni se nos dará la opción de escoger el algoritmo de descompresión (lo escogerá automáticamente el sistema). Esta opción, por lo tanto solo nos pedirá el *path* del archivo a descomprimir. Igual que al comprimir, el archivo o carpeta descomprimidos se guardarán en la misma carpeta que contiene el comprimido.

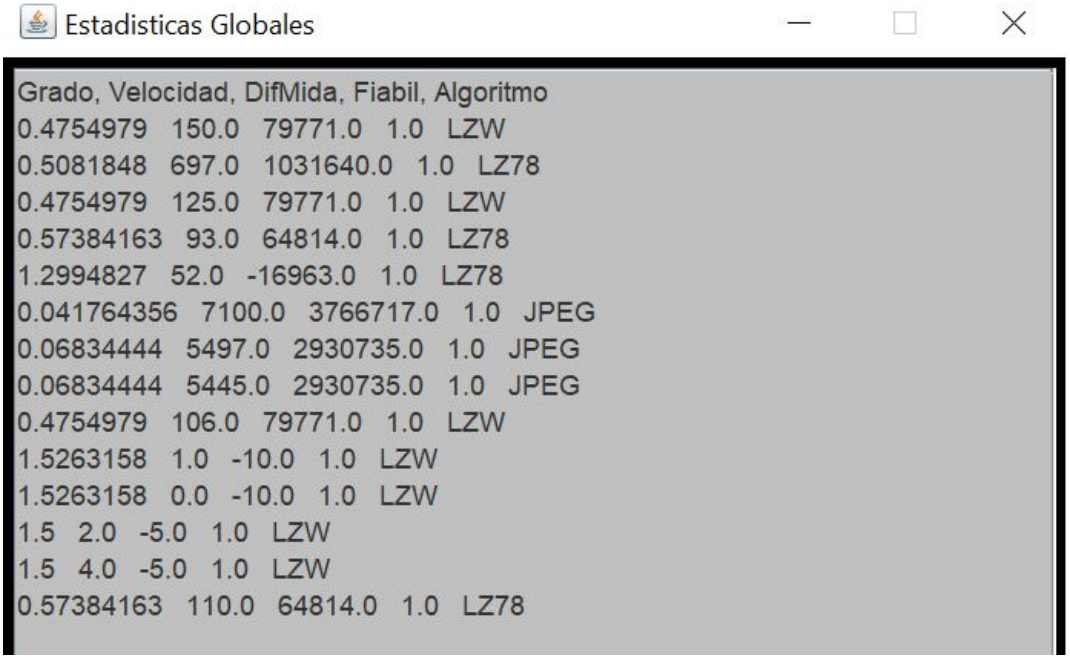
4- Estadísticas

Si seleccionamos la opción “Estadísticas”, el sistema nos preguntará qué tipo de consulta queremos realizar. Tenemos las siguientes dos opciones:



4.1- Historial de compresiones

La elección de la primera opción nos mostrará un registro de todas las compresiones que hemos realizado (la información que se nos muestra es la misma que se nos imprime después de realizar una compresión). En esta imagen se muestra qué aspecto tiene este historial.



4.2- Estadísticas de un fichero comprimido

La segunda opción nos pide que introduzcamos el *path* a un archivo con extensión ".prop" y nos muestra las estadísticas asociadas a ese fichero en particular.

7. Juegos de Prueba

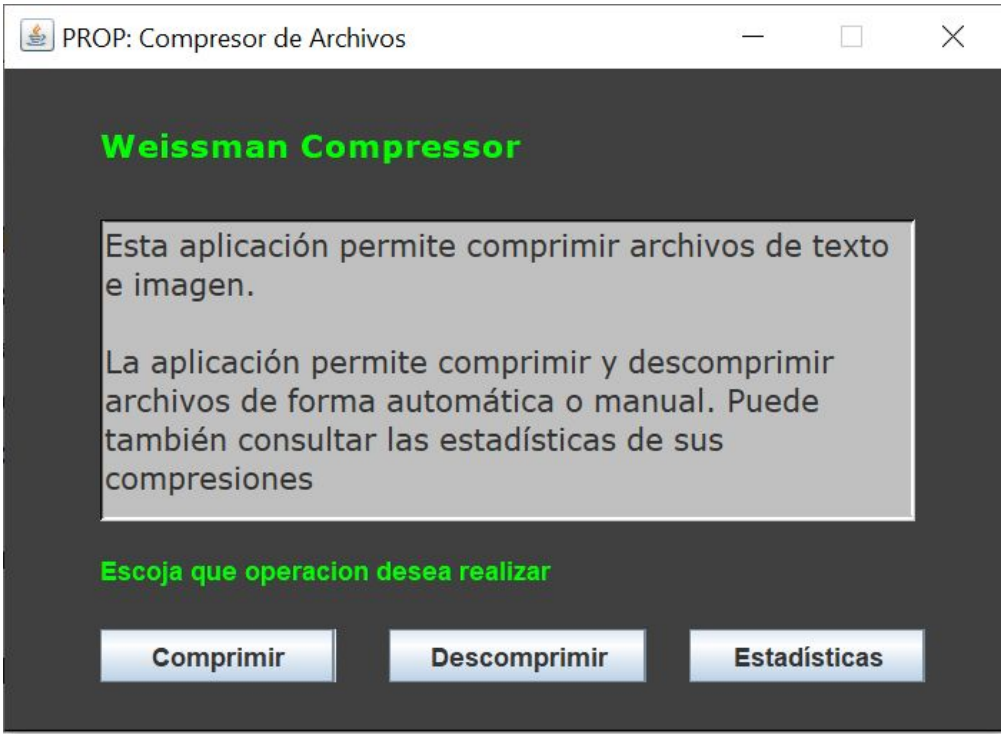
Prueba 1

Descripción: FICHERO TEXTO GRANDE
Objetivos: probar los algoritmos de compresión de textos con un fichero grande
Entrada: shakespeare.txt (archivo de 5,33 MB)
Salida: estadísticas de la compresión/descompresión + previsualización del texto
Efectos secundarios: creación del correspondiente fichero comprimido/descomprimido

La ejecución de la prueba ha sido como sigue:

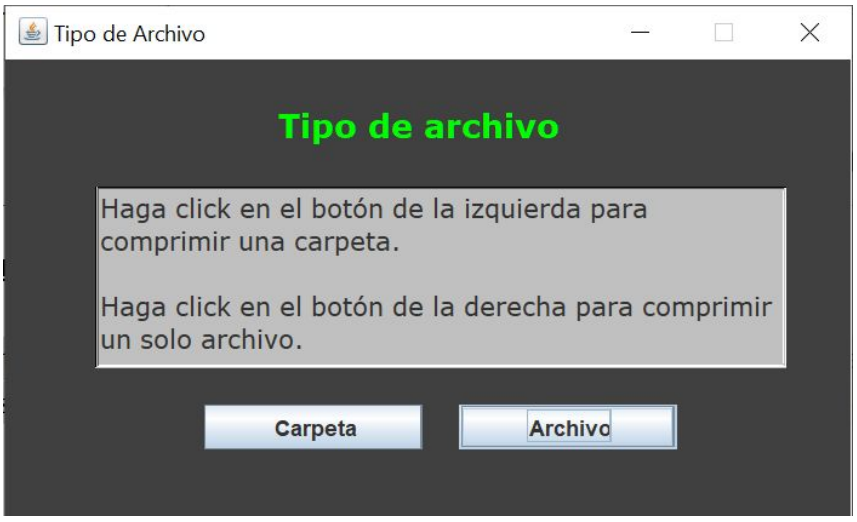
1. **Acceso a la aplicación:**

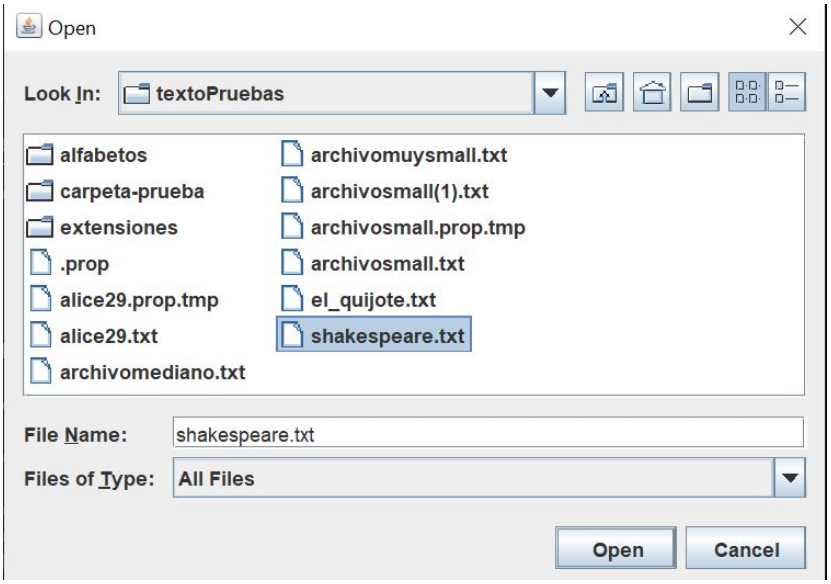
   Como se mostró anteriormente, el menú es muy sencillo. Accedemos a la aplicación y seleccionamos la opción *Comprimir*, que nos llevará al submenú de selección de tipo de archivo.



2. Seleccionando el tipo de archivo a comprimir:

Bien, como lo que queremos comprimir es solamente un archivo de texto hacemos click sobre el botón *Archivo* y nos disponemos a navegar hasta el fichero (**shakespeare.txt**) en el gestor de ficheros emergente que se despliega después de haber pulsado el botón.



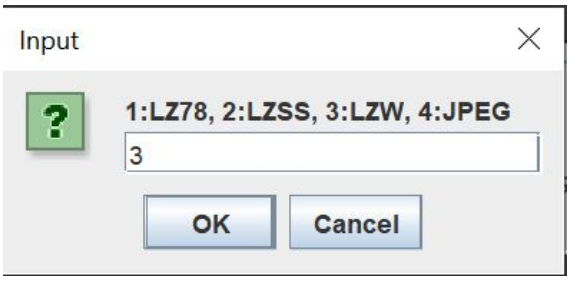


3. Seleccionando el algoritmo a emplear:

Una vez seleccionado el archivo, escojamos un algoritmo para proceder a la compresión. Primero probaremos la acción *Automática*, que escogerá el algoritmo por nosotros.

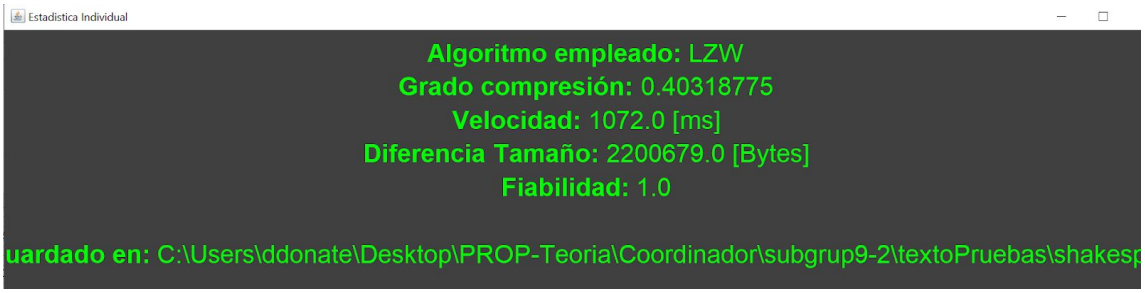
Como vemos, esta acción lleva a la generación de unas estadísticas, las que se muestran a continuación, junto con una visualización del texto que acabamos de comprimir (tras haberlo descomprimido para dicho propósito, claro).

Si en lugar del LZ78, que es el que se ha empleado, preferimos utilizar otro algoritmo, podemos hacerlo desde la selecció *Manual*. Utilicemos el LZW (la opción 3 que aparece en la entrada de texto emergente).



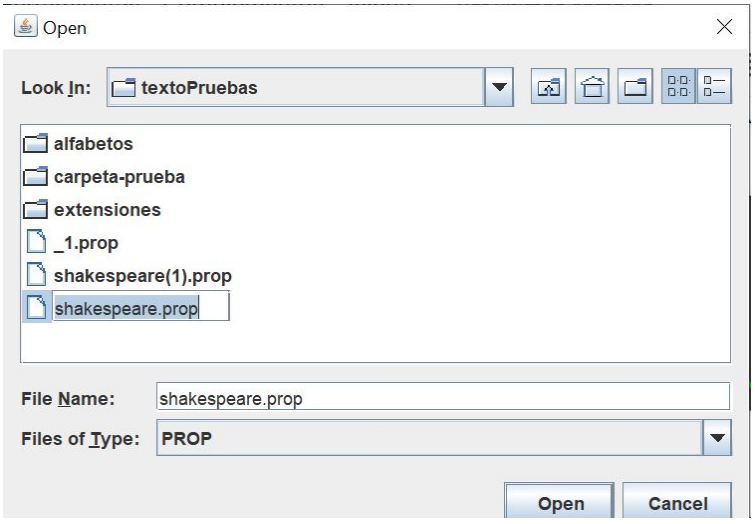
Igual que antes, se nos muestran las estadísticas asociadas a esta nueva compresión, así como la visualización del texto.

Notemos que la primera compresión ha generado un archivo llamado, **shakespeare.prop**, mientras que la segunda ha creado un **shakespeare(1).prop**.

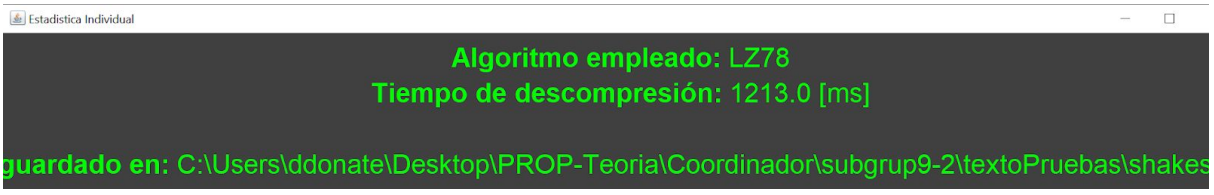


4. Descomprimiendo:

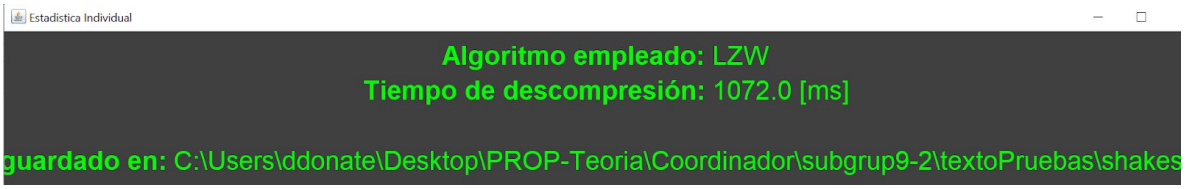
Si ahora hacemos click en el botón de *Descomprimir* (el que aparecía en el menú principal), se nos despliega el gestor de archivos para navegar hasta un fichero *.prop*. Seleccionemos el primer texto comprimido que hemos hecho generado.



Cuando lo seleccionamos se nos vuelven a mostrar estadísticas (esta vez de descompresión) y, de nuevo, una visualización del texto descomprimido.

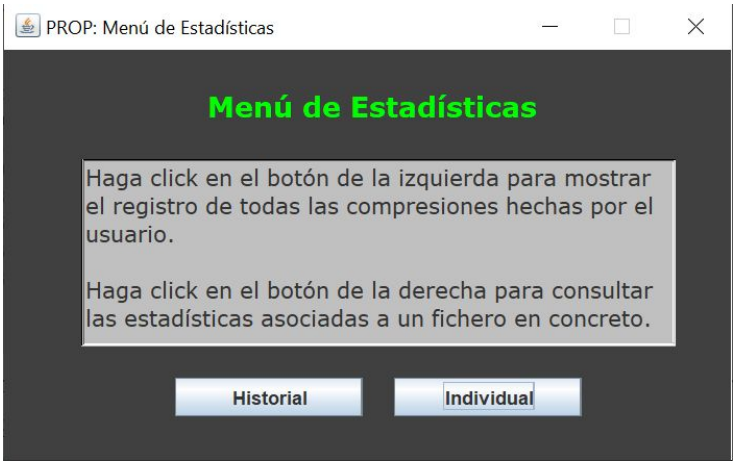


Lo propio ocurre con **shakespare(1).prop**, que se ha comprimido con el LZW.

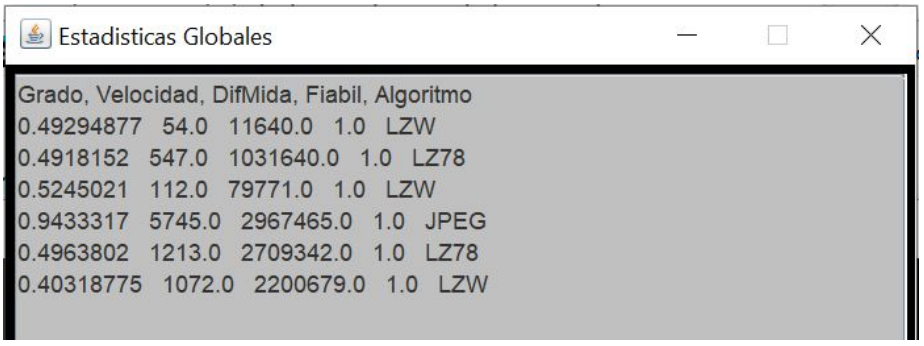


5. Consultando estadísticas:

Bien, para acabar la prueba veamos que las estadísticas se han registrado correctamente en nuestro sistema.

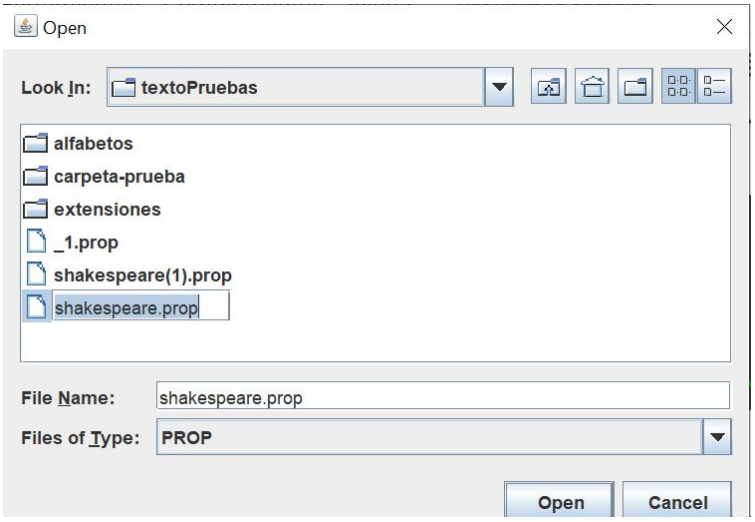


Consultemos primero el *Historial* de estadísticas, dentro de la opción *Estadísticas*, del menú principal.



Como esperábamos, las dos últimas compresiones son las que acabamos de realizar (con la correspondiente información que pertoca).

Consultemos por último la opción *Individual*, para verificar que la información que se nos muestra sobre el archivo **shakespeare.prop** es la correcta.



Igual que antes, navegamos primero hasta seleccionar el archivo.

Y, bien, observamos que los resultados coinciden con los de antes.

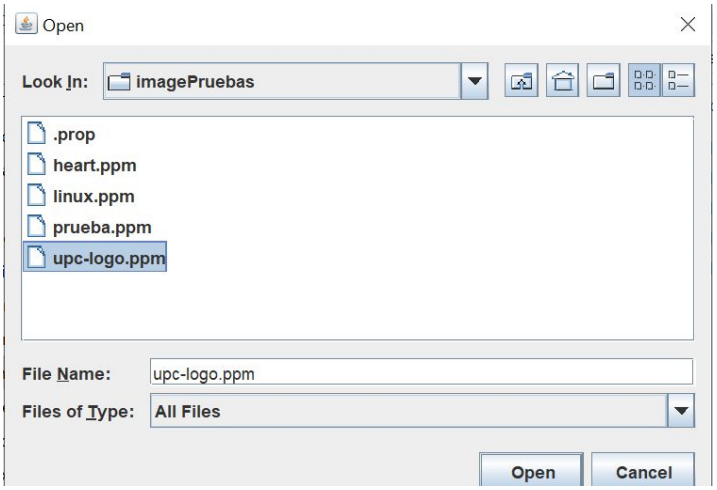
***Nota:** las siguientes pruebas no van a ser tan exhaustivas en lo que respecta a la exploración de funcionalidades del sistema. La sección de estadísticas, más concretamente, no va a mostrarse como se ha hecho en el punto 5.

Prueba 2

Descripción: FICHERO IMAGEN PEQUEÑO
Objetivos: probar el algoritmo de compresión y descompresión de imágenes de un fichero
Entrada: upc-logo.ppm (archivo de 3,07 MB)
Salida: estadísticas de la compresión/descompresión
Efectos secundarios: creación del correspondiente fichero comprimido/descomprimido

La ejecución de la prueba ha sido como sigue:

1. **Acceso a la aplicación y seleción de imagen:**
   Esta vez vamos a escoger una imagen. Los primeros pasos son idénticos a los del caso anterior. Es decir, navegamos por *Comprimir* > *Archivo* y después buscamos nuestra imagen para llevarla a comprimir.

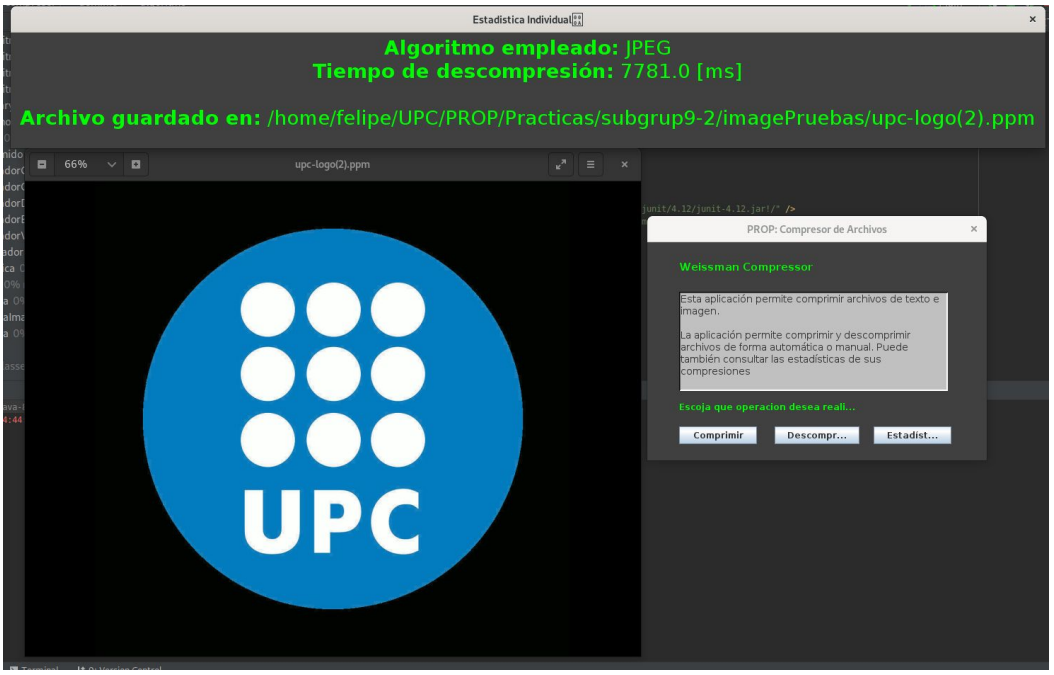


   Como vemos, hemos seleccionado el archivo **upc-logo.ppm**.

2. **Compresión de la imagen:**
   Seleccionemos ahora la opción *Automática* (alternativamente, podríamos escoger *Manual*, y seleccionar desde allí la opción *JPEG*, aunque no tiene demasiado sentido).

3. Descompresión de la imagen:

Una vez finalizado el proceso de compresión, procedamos a descomprimir el archivo.



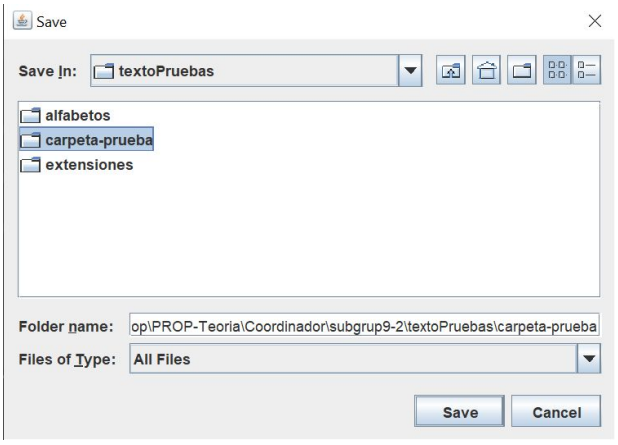
Aquí observamos la generación de estadísticas, así como la visualización de la imagen descomprimida.

Prueba 3

Descripción: CARPETA con poco contenido
Objetivos: probar los algoritmos de compresión y descompresión de carpetas con una
Entrada: carpeta-prueba (carpeta con dos ficheros: alice29.txt y archivomediano.txt, de 148 KB y 2 MB, respectivamente)
Salida: estadísticas de la compresión/descompresión
Efectos secundarios: creación de la correspondiente carpeta comprimida/descomprimida

La ejecución de la prueba ha sido como sigue:

1. **<u>Acceso a la aplicación y seleción de imagen</u>:**
   En el último test vamos a escoger una carpeta. Esta vez la ejecución de elecciones a seguir es *Comprimir* > *Carpeta* y después buscamos nuestra carpeta para llevarla a comprimir.



   Como comentamos en el manual, la compresión de carpetas produce una selección automática de los algoritmos. Así que vamos a ver la descompresión.

2. **<u>Descompresión de la carpeta</u>:**
   Una vez finalizado el proceso de compresión, procedamos a descomprimir la carpeta.