Índice

Especificación Detallada

Main

Descripción: Interficie con la que el usuario interactua para elegir el algoritmo y denotar el fichero que se quiere comprimir/descomprimir y el archivo de destino.

Atributos

- `private static String banner`
  - **Descripción**: String que contiene el *banner* que se presenta al ejecutar el Main.
- `private static Scanner scanner`
  - **Descripción**: Scanner utilizado para leer los *inputs* del usuario.

Métodos

- `public static int prompt(String[] options)`
  - **Descripción**: Dadas una opciones las imprime y da a escoger una de ellas.

    - **Return**: Devuelve la opción elegida por el usuario.
- `private static void comprimir()`
    - **Descripción**: Da a elegir al usuario el algoritmo para la compresión y le pide el nombre del archivo a comprimir y el nombre del archivo comprimido que luego pasa al CtrlDomini.
    - **Return**: Es void por tanto no devuelve nada.
- `private static void descomprimir()`
    - **Descripción**: Le pide al usuario el nombre del archivo comprimido y del archivo destino y se los pasa a CtrlDomini.
    - **Return**: Es void por tanto no devuelve nada.

CtrlDomini

Descripción: Controlador del dominio.

Atributos

- `public enum Alg {LZ78d, LZSSd, LZWd, JPEGd}`
    - **Descripción**: *Enum* usado para identificar cada algoritmo.

Métodos

- `public static Statistics compress(Alg alg, String fileIn, String fileOut, Short quality)`
    - **Descripción**: Dado un algoritmo, el nombre fichero de entrada y el nombre del fichero comprimido, ejecuta la compresión con el algoritmo pertienente.
    - **Parametros**:
        - alg: instancia de Alg que indica el algoritmo elegido para comprimir.
        - quality: calidad de compresión para el JPEG (entre 1 y 100).
    - **Return**: Devuelve las estadísticas generadas para la compresión.
- `public static Statistics decompress(String fileIn, String fileOut)`
    - **Descripción**: Dado un archivo comprimido y el nombre para el archivo descomprimido, descomprime el archivo usando el mismo algoritmo con el que se comprimió.
    - **Return**: Devuelve las estadísticas de la descompresión.

CtrlPresentacio

Descripción: Presenta datos al usuario.

Métodos

- `public static void printStatsCompress(Statistics stats)`
    - **Descripción**: Imprime las estadisticas de compresión.
    - **Return**: Es void por tanto no devuelve nada.
- `public static void printStatsDecompress(Statistics stats)`
    - **Descripción**: Imprime las estadisticas de descompresión.
    - **Return**: Es void por tanto no devuelve nada.
- `public static String readableFileSize(double d)`
    - **Descripción**: Da el formato correcto al tamaño de un fichero (B,kB,etc)
    - **Parametros**:
        - d: Tamaño de un fichero en bytes.
    - **Return**: Devuelve el tamaño del fichero en una magnitud más legible.

LZSS

Descripción: Compresión y decompresión de archivos de texto con LZSS

Atributos

- `final static int MAX_SIZE_SW`

    - **Descripción**: Tamaño máximo de la ventana corrediza.
- `final static int MAX_LENGTH_COINCIDENCE`
    - **Descripción**: Longitud máxima de una coincidencia.
- `public final static byte MAGIC_BYTE`
    - **Descripción**: Magic byte que identifica al compressor para saber con que algoritmo debemos descomprimir.
- `final static ArrayList<Character> slidingWindow = new ArrayList<Character>();`
    - **Descripción**: Contiene como mucho los últimos MAX_SIZE_SW carácteres leidos del fichero y és donde buscamos coincidencias.
- `final static ArrayList<Character> actualCharacters = new ArrayList<Character>()`
    - **Descripción**: estructura de datos que contiene los carácteres que vamos leyendo y comprobamos si esta secuencia ocurre en la ventana corrediza.

Métodos

- `public static void compress(final String inputFilename, final String outputFilename)`
    - **Descripción**: Crea el objeto lector y escritor para la compresión y llama al compresor
    - **Return**: Es void por tanto no devuelve nada.
- `private static void compress(final IO.Char.reader input, final IO.Bit.writer output)`
    - **Descripción**: Usando el algoritmo LZSS esta función comprime un archivo de texto.
    - **Parametros**:
        - input: objeto de lectura del archivo que se quiere comprimir.
        - output: objeto de escritura al archivo comprimido.
    - **Return:** Es void por tanto no devuelve nada.
- `public static void decompress(final String inputFilename, final String outputFilename)`
    - **Descripción**: Crea el objeto lector y escritor para la descompresión y llama al descompresor
    - **Return**: Es void por tanto no devuelve nada.
- `private static void decompress(final IO.Bit.reader input, final IO.Char.writer output)`
    - **Descripción**: Usando el algoritmo LZSS esta función descomprime un archivo previamente comprimido con este algoritmo.
    - **Parametros**:
        - input: objeto de lectura del archivo comprimido.
        - output: objeto de escritura al archivo descomprimido.
    - **Return:** Es void por tanto no devuelve nada.
- `private static void computeLPSArray(final int[] lps)`
    - **Descripción**: Calcula el vector *lps*, que para cada posición del vector nos dice la longitud máxima del prefijo que también es sufijo hasta esa posición.
    - **Parametros**:
        - lps: es un vector vacio que tras ejecutar esta función contiene para cada posición del vector la longitud máxima del prefijo que también es sufijo desde el principio hasta esa posición.
    - **Return:** Es void por tanto no devuelve nada.
- `private static int kmp()`
    - **Descripción**: Usando el algoritmo Knuth-Morris-Pratt calcula índice de la primera ocurrencia de un patrón.
    - **Return:** Devuelve el indice, empezando por el final, de la primera ocurrencia de actualCharacters dentro de slidingWindow o -1 si actualCharacters no se encuentra dentro del slidingWindow.
- `private static double log2(final double n)`
    - **Descripción**: Calcula el logaritmo en base 2 de un real.
    - **Return**: Devuelve el logaritmo en base 2 de n.

LZ78

Descripción: Interficie con la que el usuario interactua para elegir el algoritmo y denotar el fichero que se quiere comprimir/descomprimir y el archivo de destino.

Atributos

- `private static HashMap<String, Integer> compress_dict = new HashMap<String, Integer>()`
  - **Descripción**: Diccionario usado para la compresión.
- `private static HashMap<Integer, String> decompress_dict = new HashMap<Integer, String>()`
  - **Descripción**: Diccionario usando en la descompresión.
- `public final static byte MAGIC_BYTE`
  - **Descripción**: Magic byte que identifica al compressor para saber con que algoritmo debemos descomprimir.

Métodos

- `public static void compress(final String inputFilename, final String outputFilename)`
  - **Descripción**: Llama al metodo compress de la clase LZ78 y le pasa como parametros inputFilename y outputFilename.
  - **Return**: Es void por tanto no devuelve nada.
- `private static void compress(final IO.Char.reader input, final IO.Bit.writer output)`
  - **Descripción**: Comprime el archivo pasado por parametro input y escribe la compresion en el parametro output.
  - **Parametros**:
    - input: objeto de lectura del archivo que se quiere comprimir.
    - output: objeto de escritura al archivo comprimido.
  - **Return:** Es void por tanto no devuelve nada.
- `private static int bits_needed(final int n)`
  - **Descripción**: Calcula el numero de bits necesarios para codificar en base 2 el int pasado por parametro.
  - **Return**: Devuelve el numero de bits necesarios para codificar en base 2 el int pasado por parametro.
- `public static void decompress(final String inputFilename, final String outputFilename)`
  - **Descripción**: Llama al metodo decompress de la clase LZ78 y le pasa como parametros inputFilename y outputFilename.
  - **Return**: Es void por tanto no devuelve nada.
- `private static void decompress(final IO.Bit.reader input, final IO.Char.writer output)`
  - **Descripción**: Descomprime el archivo comprimido pasado por input y escribe la descompresion por el parametro output.
  - **Parametros**:
    - input: objeto de lectura del archivo comprimido.
    - output: objeto de escritura al archivo descomprimido.
  - **Return:** Es void por tanto no devuelve nada.

LZW

Descripción: Compresión y descompresión de archivos de texto con LZW.

Atributos

- `public final static byte MAGIC_BYTE`
  - **Descripción**: Magic byte que identifica al compressor para saber con que algoritmo debemos descomprimir.
- `private final static int DICTIONARY_SIZE`
  - **Descripción**: Tamaño inicial del diccionario.
- `private static HashMap<String, Integer> compressionDictionary`
  - **Descripción**: Diccionario de compresión.
- `private static HashMap<Integer, String> decompressionDictionary`
  - **Descripción**: Diccionario usado en la descompresión

Métodos

- `private static void createCompressionDictionary()`
  - **Descripción**: Crea el diccionario de compresión y lo inicializa.

    - **Return**: Es void por tanto no devuelve nada.
- `private static void createDecompressionDictionary()`
    - **Descripción**: Crea el diccionario de descompresión y lo inicializa.
    - **Return**: Es void por tanto no devuelve nada.
- `public static void compress(final String inputFilename, final String outputFilename)`
    - **Descripción**: Llama a una función que comprime un archivo de texto.
    - **Return**: Es void por tanto no devuelve nada.
- `private static void compress (IO.Char.reader input, IO.Bit.writer output)`
    - **Descripción**: Comprime un archivo de texto implementando un algoritmo LZW.
    - **Parametros**:
        - input: Objeto de lectura del archivo que se quiere comprimir.
        - output: Objeto de escritura al archivo comprimido.
    - **Return:** Es void por tanto no devuelve nada.
- `public static void decompress(final String inputFilename, final String outputFilename)`
    - **Descripción**: LLama a una función que descomprime un archivo de texto.
    - **Return**: Es void por tanto no devuelve nada.
- `private static void decompress (IO.Bit.reader input, IO.Char.writer output)`
    - **Descripción**: Comprime un archivo de texto implementando un algoritmo LZW.
    - **Parametros**:
        - input: Objeto de lectura del archivo que se quiere descomprimir.
        - output: Objeto de ecritura del archivo desccomprimido.
    - **Return:** Es void por tanto no devuelve nada.

JPEG

Descripción: Compresión y descompresión de imágenes PPM con JPEG.

Atributos

- `public final static byte MAGIC_BYTE`
    - **Descripción**: magic byte del JPEG.

Métodos

- `public static void compress(final String inputFile, final String outputFile, final short quality)`
    - **Descripción**: Comprime una imagen PPM bloque a bloque.
    - **Parametros**:
        - quality: Calidad de compresión (1-100) donde 100 es la mejor calidad.
    - **Return**: Es void por tanto no devuelve nada.
- `public static void decompress(final String inputFile, final String outputFile)`
    - **Descripción**: Descomprime un fichero comprimido en JPEG y lo guarda la imagen resultante en un fichero PPM raw.
    - **Return**: Es void por tanto no devuelve nada.
- `public static short[] readBlock(final Huffman huffAC, final Huffman huffDC, final IO.Bit.reader file)`
    - **Descripción**: Lee un bloque codificado con las tablas Huffman.
    - **Parametros**:
        - huffAC: Tabla Huffman de valores AC.
        - huffDC: Tabla Huffman de valores DC.
        - file: Fichero comprimido del que leer.
    - **Return**: Devuelve un bloque codificado sin Huffman.
- `private static short readHuffman(Huffman huff, IO.Bit.reader file)`
    - **Descripción**: Lee un código Huffman del fichero y lo decodifica.
    - **Parametros**:
        - huff: Tabla Huffman.
    - **Return**: Devuelve el código Huffman decodificado.

- `private static void write(int value, int l, IO.Bit.writer file)`
  - **Descripción**: Escribe en un fichero un valor en binario, si value es positivo escribe value y si es negativo escribe ~(-value).
  - **Parametros**:
    - value: valor a escribir
    - l: número de bits con los que codificar value.
    - file: fichero donde escribir value.
  - **Return**: Es void por tanto no devuelve nada.
- `private static short read(int length, IO.Bit.reader file)`
  - **Descripción**: Lee de un fichero un valor en binario de longitud l, si es un valor positivo lo lee tal cual y si es negativo lo lee como -(valor).
  - **Parametros**:
    - l: número de bits con los que codificar value.
    - file: fichero donde escribir value.
  - **Return**: Es void por tanto no devuelve nada.

JPEGBlock

Descripción: Codifica y decodifica bloques 8x8 con JPEG.

Métodos

- `private static int bitLength(final short n)`
  - **Descripción**: Número de bits necesarios para representar el short n.
  - **Return**: Devuelve el número de bits necesarios para representar n.
- `public static short[] encode(final short quality, final boolean isChrominance, final byte[][] data)`
  - **Descripción**: Comprime un bloque 8x8 aplicando DCT, cuantización, zigZag y RLE.
  - **Parametros**:
    - quality: Calidad de compresión (1-100).
    - isChrominance: Si es un bloque de Chrominance (si falso Luminance).
    - data: Bloque 8x8 a codificar.
  - **Return**: Devuelve el bloque 8x8 codificado en RLE.
- `public static byte[][] decode(final short quality, final boolean isChrominance, final short[] data)`
  - **Descripción**: Deshace RLE, zigZag, quantización y DCT para obtener el bloque 8x8 original.
  - **Parametros**:
    - quality: Calidad de compresión (1-100).
    - isChrominance: Si es un bloque de Chrominance (si falso Luminance).
    - data: Bloque 8x8 a codificado en RLE.
  - **Return**: Devuelve el bloque 8x8 decodificado.

Subclase: DCT

Descripción: Codificación y decodificación del paso *Discrete Cosine Transform* de la compresión/descompresión JPEG para un bloque de 8x8.

Subclase: Quantizacion

Descripción: Codificación y decodificación del paso de cuantización con tablas predefinidas, ajustadas según la calidad de compresión especificada, para la compresión/descompresión JPEG de un bloque de 8x8.

Subclase: ZigZag

Descripción: Aplasta un bloque 8x8 bytes en zigZag.

Subclase: RLE

Descripción: Codifica/Decodifica en RLE (Run Length Encoding).

PpmImage

Descripción: Representa una imagen PPM.

Atributos

- `private byte[][][] pixels`
  - **Descripción**: Representa la imagen en 3 canales (RGB o YCbCr) con dos dimensiones por canal.
- `private int width`
  - **Descripción**: Almacena la anchura de la imagen.
- `private int height`
  - **Descripción**: Guarda la altura de la imagen.

Métodos

- `public void setDimensions(final int w, final int h)`:
  - **Descripción**: Inicializa la imagen vacía con los valores de anchura y altura.
  - **Return**: Es void por tanto no devuelve nada.
- `public void readFile(final String filename)`
  - **Descripción**: Lee una fichero imagen y lo guarda en la memoria.
  - **Return**: Es void por tanto no devuelve nada.
- `public void writeFile(final String filename)`
  - **Descripción**: Escribe la imagen en un fichero.
  - **Return**: Es void por tanto no devuelve nada.
- `private int readInt(final IO.Byte.reader file)`
  - **Descripción**: Lee el siguiente entero codificado en ASCII que encuentra en el fichero.
  - **Return**: Devuelve el entero leído.
- `private byte doubleToByte(final double d)`
  - **Descripción**: Convierte un double a byte.
  - **Return**: Retorna un byte.
- `public void toRGB()`
  - **Descripción**: Convierte la imagen de espacio de color YCbCr a espacio RGB.
  - **Return**: Es void por tanto no devuelve nada.
- `public void toYCbCr()`
  - **Descripción**: Convierte la imagen de espacio de color RGB a espacio YCbCr.
  - **Return**: Es void por tanto no devuelve nada.
- `public byte[][] readBlock(final int channel, final int x, final int y)`
  - **Descripción**: Devuelve un bloque de la imagen de 8x8.
  - **Parametros**
    - channel: Canal de color del bloque.
    - x,y: Posición del bloque.
  - **Return**: Devuelve el bloque 8x8 bytes.
- `public void writeBlock(final byte[][] block, final int channel, final int x, final int y)`
  - **Descripción**: Escribe un bloque de la imagen de 8x8.
  - **Parametros**
    - block: Bloque de 8x8 bytes a escribir.
    - channel: Canal de color del bloque.
    - x,y: Posición del bloque.
  - **Return**: Es void por tanto no devuelve nada.
- `public int width()`
  - **Descripción**: Consultora de *width*.
  - **Return**: Devuelve la anchura de la imagen en pixels.
- `public int height()`
  - **Descripción**: Consultora de *height*.

    - **Return**: Devuelve la altura de la imagen en pixels.
- `public int columns()`
    - **Descripción**: Consultora del número de bloques 8x8 que caben horizontalmente en la imagen.
    - **Return**: Retorna la anchura de la imagen en bloques de 8x8 bytes.
- `public int rows()`
    - **Descripción**: Consultora del número de bloques 8x8 que caben verticalmente en la imagen.
    - **Return**: Retorna la altura de la imagen en bloques de 8x8 bytes.

Subclase: InvalidFileFormat

Descripción: Extiende la classe Exception y se lanza cuando el formato de la imagen no es un PPM válido.

IO

Descripción: Classes IO para lectura y escritura de Char, Byte, y Bit con buffer.

Subclase: Char

Descripción: Lector/escritor char a char con buffer.

Subclase: reader **Descripción**: Extiende la clase *BufferedReader* para poder construir un buffer de lectura char a char a partir del archivo que recibe.

Subclase: writer **Descripción**: Extiende la clase *BufferedWriter* para poder construir un buffer de escritura char a char a partir del archivo en el que quiere escribir.

Subclase: Byte

Descripción: Lector/escritor byte a byte con buffer.

Subclase: reader **Descripción**: Extiende la clase *BufferedInputStream* para poder construir un buffer de lectura byte a byte a partir del archivo que recibe.

Subclase: writer **Descripción**: Extiende la clase *BufferedOutputStream* para poder construir un buffer de escritura byte a byte a partir del archivo en el que quiere escribir.

Subclase: Bit

Descripción: Lector/escritor bit a bit con buffer.

Subclase: reader **Descripción**: Permite leer bit a bit un fichero. Además incorpora la posibilidad de leer también bytes, chars, ints e incluso BitSetL.

Atributos

- `private final BufferedInputStream in`
    - **Descripción**: *BufferedInputStream* a través del cual leemos.
- `private int buffer`
    - **Descripción**: Buffer de 8 bits donde se van almacenando los bits leidos, ya que solo podemos leer un byte como mínimo.
- `private int n`
    - **Descripción**: Número de bits almacenados en el buffer.

Métodos

- `public reader(final String filename)`
  - **Descripción**: Constructora que incializa el *BufferedInputStream* para poder leer del fichero y inicializa también los atributos n y buffer.
  - **Return**: Es void por tanto no devuelve nada
- `private void fill()`
  - **Descripción**: Lee un byte del fichero, unidad mínima que se puede leer, y lo almacena en el buffer para que pueda ser leido bit a bit. También actualiza el valor de n a 8.
- `public boolean read()`
  - **Descripción**: Si quedan bits en el buffer retorna el siguient bit, sinó rellena el buffer leyendo un byte del fichero y retorna el bit de mayor peso.
  - **Return**: Retorna el siguiente bit leido del fichero.
- Existen varios *read[Byte/Char/Int/BitSetL]()* que leen del archivo un byte, char, int y BitSetL y devuelven el byte, char, int y BitSetL respectivamente. Además en el caso del BitSetL recibe como parametro cuantos bits se quieren leer.
- `private int readMask(int mask)`
  - **Descripción**: Lee *mask* bits del fichero.
  - **Return**: Retorna los *mask* bits leidos del archivo interpretados como entero.
- `public void close()`
  - **Descripción**: Invocado al finalizar la lectura para cerrar el *BufferedInputStream*.
  - **Return**: Es void por tanto no devuelve nada

Subclase: writer **Descripción**: Permite escribir bit a bit a un fichero. Además implementa la posibilidad de escribir también bytes, chars, ints y BitSetL.

Atributos

- `private final BufferedOutputStream out`
  - **Descripción**: *BufferedOutputStream* a través del cual escribimos.
- `private int buffer`
  - **Descripción**: Buffer de 8 bits donde se van almacenando los bits que se quieren escribir.
- `private int n`
  - **Descripción**: Número de bits almacenados en el buffer.

Métodos

- `public writer(final String filename)`
  - **Descripción**: Constructora que incializa el *BufferedOutputStream* para que abra un fichero al que queremos escribir y inicializa también los atributos n y buffer.
  - **Return**: Es void por tanto no devuelve nada.
- `public void write(final boolean bit)`
  - **Descripción**: Escribe un bit en buffer, cuando el buffer esta lleno (un byte) se escribe por *out*.
  - **Return**: Es void por tanto no devuelve nada.
- Tiene varios métodos `write(...)` que escriben byte, char, int y BitSetL a través de *out*. Son todas `void` y por lo tanto no retornan nada.
- `private void writeMask(final int num, int mask)`
  - **Descripción**: Escribe un entero usando *mask* bits.
  - **Return**: Es void por tanto no devuelve nada.
- `private void clear()`
  - **Descripción**: Se invoca para escribir en el fichero los bits que hay en el buffer, si faltan bits para completar el byte se rellena con ceros en los bits de menor peso.
  - **Return**: Es void por tanto no devuelve nada.
- `public void flush()`
  - **Descripción**: Usado para llamar al método `clear()` y hace un *flush* en el *BufferedOutputStream*.
  - **Return**: Es void por tanto no devuelve nada.
- `public void close()`
  - **Descripción**: Llamar a `flush()` y cierra el *BufferedOutputStream*.

- **Return**: Es void por tanto no devuelve nada.

Statistics

Descripción: Genera las estadísticas de compresión/descompresión siguientes:

- Tiempo de ejecución
- Velocidad de ejecución
- Porcentaje de compresión/descompresión
- Tamaño del fichero de entrada
- Tamaño del fichero de salida