

Juegos de pruebas

En este documento se detallarán los juegos de pruebas usados en cada driver. Se describirá para cada caso de uso de cada driver todos los escenarios y como el programa reacciona a ellos.

CtrlEstadistica	4
Prueba 1	4
Prueba 2	4
OutputAlgoritmo	6
Prueba 1	6
LZSS	6
Prueba 1	6
Prueba 2	7
Prueba 3	7
Prueba 4	8
Prueba 5	8
Prueba 6	9
Prueba 7	9
Prueba 8	10
Prueba 9	10
Prueba 10	11
Prueba 11	11
Prueba 12	12
Prueba 13	12
Prueba 14	13
LZW	14
Prueba 1	14
Prueba 2	14
Prueba 3	15
Prueba 4	15
Prueba 5	16
Prueba 6	16
Prueba 7	17
Prueba 8	17
Prueba 9	18
Prueba 10	18
Prueba 11	19
Prueba 12	19
Prueba 13	20

LZ78	21
Prueba 1	21
Prueba 2	21
Prueba 3	21
Prueba 4	22
Prueba 5	22
Prueba 6	23
Prueba 7	23
Prueba 8	24
Prueba 9	24
Prueba 10	25
Prueba 11	25
Prueba 12	25
JPEG	27
Prueba 1	27
Prueba 2	27
Prueba 3	28
Prueba 4	29
Prueba 5	29
Prueba 6	30
Trie	31
Prueba 1	31
Prueba 2	32
Prueba 3	32
Prueba 4	33
Prueba 5	34
Prueba 6	35
ProcesoComprimir	37
Prueba 1	37
Prueba 2	37
Prueba 3	38
Prueba 4	38
Prueba 5	39
Prueba 6	39
Prueba 7	40
Prueba 8	41
Prueba 9	41
ProcesoDescomprimir	43
Prueba 1	43
Prueba 2	43

Prueba 3	43
Prueba 4	44
Prueba 5	44
Prueba 6	44
Prueba 7	45
Prueba 8	45
Prueba 9	45
Prueba 10	46
Prueba 11	46
Prueba 12	47
CtrlProcesos	48
Prueba 1	48
Prueba 2	48
Prueba 3	49
Prueba 4	49
Prueba 5	50
Prueba 6	50
Prueba 7	51
Prueba 8	51
DatosProceso	53
Prueba 1	53
Prueba 2	53
Prueba 3	54
Prueba 4	55
Prueba 5	55
Prueba 6	56
Prueba 7	57

CtrlEstadistica

Prueba 1

Descripción: Probar el método de estadísticas de CtrlEstadistica con un algoritmo de los 4 disponibles.

Objetivo: Comprobar que se muestran las estadísticas generales del algoritmo escogido usando los ficheros de texto donde se guardan las estadísticas..

Driver Usado: CtrlEstadisticaDriver

Entrada: Información obtenida tras compresión y/o descompresión de archivo/s con el algoritmo escogido. Y algoritmo escogido por terminal 1 o JPEG

Resultado Esperado: Visualización de estadísticas generales del algoritmo JPEG

Salida por pantalla:

Escriba el algoritmo de compresión que quiera consultar, de entre los siguientes:

1. jpeg
2. lzss
3. lz78
4. lzw
5. salir

1

Estadísticas generales del algoritmo JPEG:

Archivos comprimidos: 2

Tiempo medio de compresión: 1679758350 ns

Porcentaje medio de tamaño respecto al original: 50.0 %

Archivos descomprimidos: 1

Tiempo medio de descompresión: 768961700 ns

Porcentaje medio de tamaño respecto al original: 196.0 %

Salida: Estadísticas generales del algoritmo JPEG.

Veredicto: OK

Prueba 2

Descripción: Probar el método de estadísticas de CtrlEstadistica con un algoritmo de los 4 disponibles.

Objetivo: Comprobar que se muestran las estadísticas generales de un algoritmo escogido que no tiene ficheros de estadísticas.

Driver Usado: CtrlEstadisticaDriver

Entrada: Algoritmo escogido por terminal 2 o LZSS.

Resultado Esperado: Visualización de estadísticas generales vacías del algoritmo LZSS

Salida por pantalla:

Escriba el algoritmo de compresión que quiera consultar, de entre los siguientes:

1. jpeg
2. lzss
3. lz78
4. lzw
5. salir

2

Estadísticas generales del algoritmo LZSS:

Archivos comprimidos: 0

Archivos descomprimidos: 0

Salida: Estadísticas generales del algoritmo LZSS vacías

Veredicto: OK

OutputAlgoritmo

Prueba 1

Descripción: Probar que se genera adecuadamente una instancia de OutputAlgoritmo.

Objetivo: Crear una instancia de OutputAlgoritmo y observar que se han asignado correctamente los archivos.

Driver Usado: OutputAlgoritmoDriver

Entrada: Introducir el número “417964” y la secuencia de bytes “aubaabCoeç83”.

Resultado Esperado: La instancia se genera correctamente y el valor de sus componentes es el mismo que los introducidos.

Salida por pantalla:

```
Escoja una opción:  
1. Crear una instancia de OutputAlgoritmo  
2. Salir  
1  
Introduzca a continuación una cifra (por ejemplo 123):  
417964  
Introduzca a continuación una tira de texto (por ejemplo asdf1234):  
aubaabCoeç83  
A continuación se creará OutputAlgoritmo con tiempo: 417964 y output: auba  bCoe  83  
Ahora se probará el valor del tiempo:  
El resultado de tiempo es: 417964  
Ahora se probará el valor del output:  
El resultado de output es: auba  bCoe  83
```

Salida: Se ha generado una instancia de OutputAlgoritmo con los atributos iguales a los introducidos.

Veredicto: OK

LZSS

Prueba 1

Descripción: Probar el método de compresión de la clase LZSS con un archivo de texto vacío.

Objetivo: Comprobar que comprime adecuadamente un fichero vacío.

Driver Usado: LZSSDriver

Entrada: empty.txt → un fichero de texto que no tiene ningún carácter.

Resultado Esperado: res Esperado.lzss → El resultado esperado de la compresión es un archivo que contenga 4 bytes delbitset, que representan un 0. Por tanto 00 00 00 00. Esta compresión no reduce el tamaño del archivo original, esto es porque no hay coincidencias en el archivo.

Salida por pantalla:

Introduzca uno de los siguientes comandos disponibles:

```
1. comprimir
2. descomprimir
3. salir
1
Escriba el path relativo de un fichero .txt a comprimir:
/JuegosDePruebas/Prueba1/empty.txt
Se inicia el proceso
El archivo /home/jan/Documents/PROP/PROPCompressor/EXE/LZSS//JuegosDePruebas/Prueba1/empty.txt se ha comprimido correctamente!
Ha tardado 4.71842E-4s y se ha guardado en /home/jan/Documents/PROP/PROPCompressor/EXE/LZSS//JuegosDePruebas/Prueba1/empty.lzss.
El cambio de tamaño pasa de 0B a 4B con diferencia de -4B que resulta en un Infinity% del archivo original.
```

Salida: empty.lzss → vemos que es un archivo que contiene 4 bytes con valor hexa 00 00 00 00.

Veredicto: OK

Prueba 2

Descripción: Probar el método de compresión de la clase LZSS con un archivo de texto que contiene un solo carácter ASCII.

Objetivo: Comprobar que comprime adecuadamente un fichero que contiene un solo carácter ASCII.

Driver Usado: LZSSDriver

Entrada: a.txt

Resultado Esperado: res Esperado.lzss → El resultado esperado de la compresión es un archivo que contenga la medida delbitset que es nulo, un byte con la "a" codificada y un endl. Esta compresión no reduce el tamaño del archivo original, esto es porque no hay coincidencias en el archivo.

Salida por pantalla:

Introduzca uno de los siguientes comandos disponibles:

```
1. comprimir
2. descomprimir
3. salir
1
Escriba el path relativo de un fichero .txt a comprimir:
/JuegosDePruebas/Prueba2/a.txt
Se inicia el proceso
El archivo /home/jan/Documents/PROP/PROPCompressor/EXE/LZSS//JuegosDePruebas/Prueba2/a.txt se ha comprimido correctamente!
Ha tardado 0.001440033s y se ha guardado en /home/jan/Documents/PROP/PROPCompressor/EXE/LZSS//JuegosDePruebas/Prueba2/a.lzss.
El cambio de tamaño pasa de 2B a 6B con diferencia de -4B que resulta en un 300.0% del archivo original.
```

Salida: a.lzss → vemos que es un archivo que contiene 4 bytes con valor hexa 00 00 00 00, y luego 61 y un endl.

Veredicto: OK

Prueba 3

Descripción: Probar el método de compresión de la clase LZSS con un archivo de texto que contiene un solo carácter ASCII Extended.

Objetivo: Comprobar que comprime adecuadamente un fichero que contiene un solo carácter ASCII Extended que por naturaleza se codifica en dos bytes y no en uno.

Driver Usado: LZSSDriver

Entrada: ASCIilexended.txt

Resultado Esperado: res Esperado.lzss → El resultado esperado de la compresión es un archivo que contenga 4 bytes delbitset, dos bytes que codifican el carácter en ASCIIextended y un endl. Esta compresión no reduce el tamaño del archivo original, esto es porque no hay coincidencias en el archivo.

Salida por pantalla:

Introduzca uno de los siguientes comandos disponibles:

```
1. comprimir
2. descomprimir
3. salir
1
Escriba el path relativo de un fichero .txt a comprimir:
/JuegosDePruebas/Prueba3/ASCIIextended.txt
Se inicia el proceso
El archivo /home/jan/Documents/PROP/PROPCompressor/EXE/LZSS//JuegosDePruebas/Prueba3/ASCIIextended.txt se ha comprimido correctamente!
Ha tardado 8.8263E-5s y se ha guardado en /home/jan/Documents/PROP/PROPCompressor/EXE/LZSS//JuegosDePruebas/Prueba3/ASCIIextended.lzss.
El cambio de tamaño pasa de 3B a 7B con diferencia de -4B que resulta en un 233.0% del archivo original.
```

Salida: ASCIIextended.lzss → vemos que es un archivo que contiene 4 bytes con valor hexa 00 00 00 00, y luego dos bytes (C3 A7) que codifican “ç” y un endl.

Veredicto: OK

Prueba 4

Descripción: Probar el método de compresión de la clase LZSS con un archivo de texto que contiene una serie de caracteres ASCII.

Objetivo: Comprobar que comprime adecuadamente un fichero que contiene una serie de caracteres ASCII.

Driver Usado: LZSSDriver

Entrada: abc.txt → fichero que contiene 44 bytes que son caracteres ASCII.

Resultado Esperado: res Esperado.lzss → El resultado esperado de la compresión es un archivo que contenga 4 bytes delbitset que codifican un 0 ya que no hay coincidencias y elbitset es de tamaño 0 y los bytes del archivo original. Esta compresión no reduce el tamaño del archivo original, esto es porque no hay coincidencias en el archivo.

Salida por pantalla:

Introduzca uno de los siguientes comandos disponibles:

```
1. comprimir
2. descomprimir
3. salir
1
Escriba el path relativo de un fichero .txt a comprimir:
/JuegosDePruebas/Prueba4/abc.txt
Se inicia el proceso
El archivo /home/jan/Documents/PROP/PROPCompressor/EXE/LZSS//JuegosDePruebas/Prueba4/abc.txt se ha comprimido correctamente!
Ha tardado 4.97103E-4s y se ha guardado en /home/jan/Documents/PROP/PROPCompressor/EXE/LZSS//JuegosDePruebas/Prueba4/abc.lzss.
El cambio de tamaño pasa de 44B a 48B con diferencia de -4B que resulta en un 109.0% del archivo original.
```

Salida: abc.lzss → vemos que es un archivo que contiene 4 bytes con valor hexa 00 00 00 00, y luego los bytes del archivo original.

Veredicto: OK

Prueba 5

Descripción: Probar el método de compresión de la clase LZSS con un archivo de texto que contiene una serie de caracteres ASCII Extended.

Objetivo: Comprobar que comprime adecuadamente un fichero que contiene una serie de caracteres ASCII Extended que por naturaleza se codifica en dos bytes y no en uno.

Driver Usado: LZSSDriver

Entrada: abcASCIIextended.txt

Resultado Esperado: res Esperado.lzss → El resultado esperado de la compresión es un archivo que contenga 4 bytes delbitset, los bytes que codifican los caracteres en ASCIIextended y un endl. Esta compresión no reduce el tamaño del archivo original, esto es porque no hay coincidencias en el archivo.

Salida por pantalla:

Introduzca uno de los siguientes comandos disponibles:

1. comprimir
2. descomprimir
3. salir

1

Escriba el path relativo de un fichero .txt a comprimir:

/JuegosDePruebas/Prueba5/abcASCIIextended.txt

Se inicia el proceso

El archivo /home/jan/Documents/PROP/PROPCompressor/EXE/LZSS//JuegosDePruebas/Prueba5/abcASCIIextended.txt se ha comprimido correctamente!
Ha tardado 3.47057E-4s y se ha guardado en /home/jan/Documents/PROP/PROPCompressor/EXE/LZSS//JuegosDePruebas/Prueba5/abcASCIIextended.lzss.
El cambio de tamaño pasa de 27B a 31B con diferencia de -4B que resulta en un 114.0% del archivo original.

Salida: ASCIIextended.lzss → vemos que es un archivo que contiene 4 bytes delbitset, los bytes que codifican los caracteres en ASCIIextended y un endl

Veredicto: OK

Prueba 6

Descripción: Probar el método de compresión de la clase LZSS con un archivo de texto que contiene una serie combinada de caracteres ASCII y ASCII Extended.

Objetivo: Comprobar que comprime adecuadamente un fichero que contiene una serie combinada de caracteres ASCII y ASCII Extended. Esta compresión no reduce el tamaño del archivo original, esto es porque no hay coincidencias en el archivo.

Driver Usado: LZSSDriver

Entrada: ASCIIplusASCIIextended.txt

Resultado Esperado: res Esperado.lzss → El resultado esperado de la compresión es un archivo que contenga 4 bytes delbitset, los bytes que codifican los caracteres en ASCIIextended y ASCII y un endl.

Salida por pantalla:

Introduzca uno de los siguientes comandos disponibles:

1. comprimir
2. descomprimir
3. salir

1

Escriba el path relativo de un fichero .txt a comprimir:

/JuegosDePruebas/Prueba6/ASCIIplusASCIIextended.txt

Se inicia el proceso

El archivo /home/jan/Documents/PROP/PROPCompressor/EXE/LZSS//JuegosDePruebas/Prueba6/ASCIIplusASCIIextended.txt se ha comprimido correctamente!
Ha tardado 2.32667E-4s y se ha guardado en /home/jan/Documents/PROP/PROPCompressor/EXE/LZSS//JuegosDePruebas/Prueba6/ASCIIplusASCIIextended.lzss.
El cambio de tamaño pasa de 28B a 32B con diferencia de -4B que resulta en un 114.0% del archivo original.

Salida: ASCIIplusASCIIextended.lzss → vemos que es un archivo que contiene 4 bytes delbitset, los bytes que codifican los caracteres ASCII y ASCIIextended y un endl.

Veredicto: OK

Prueba 7

Descripción: Probar el método de compresión de la clase LZSS con un archivo de texto que contiene una serie de caracteres ASCII.

Objetivo: Comprobar que comprime adecuadamente un fichero que contiene una serie de caracteres ASCII.

Driver Usado: LZSSDriver

Entrada: cans.txt → fichero que contiene 43 bytes que son caracteres ASCII.

Resultado Esperado: res Esperado.lzss → El resultado esperado de la compresión es un archivo que contenga 4 bytes delbitset, en este caso elbitset va a continuación porque como ya hay coincidencias estas se codifican y los flags están en elbitset. A estebitset le sigue la codificación de los caracteres.

Salida por pantalla:

Introduzca uno de los siguientes comandos disponibles:

```
1. comprimir
2. descomprimir
3. salir
1
Escriba el path relativo de un fichero .txt a comprimir:
/JuegosDePruebas/Prueba7/cans.txt
Se inicia el proceso
El archivo /home/jan/Documents/PROP/PROPCompressor/EXE/LZSS//JuegosDePruebas/Prueba7/cans.txt se ha comprimido correctamente!
Ha tardado 4.58064E-4s y se ha guardado en /home/jan/Documents/PROP/PROPCompressor/EXE/LZSS//JuegosDePruebas/Prueba7/cans.lzss.
El cambio de tamaño pasa de 43B a 35B con diferencia de 8B que resulta en un 81.0% del archivo original.
```

Salida: cans.lzss → vemos que es un archivo que contiene 4 bytes que codifican el tamaño delbitset, seguido delbitset y luego los caracteres codificados.

Veredicto: OK

Prueba 8

Descripción: Probar el método de compresión de la clase LZSS con un archivo de texto que contiene una serie de caracteres ASCII y ASCIilexended.

Objetivo: Comprobar que comprime adecuadamente un fichero que contiene una serie de caracteres ASCII y ASCIilexended.

Driver Usado: LZSSDriver

Entrada: cansExtended.txt → fichero que contiene 138 bytes que son caracteres ASCII y ASCIilexended.

Resultado Esperado: res Esperado.lzss → El resultado esperado de la compresión es un archivo que contenga 4 bytes delbitset, en este caso elbitset va a continuación porque como ya hay coincidencias estas se codifican y los flags están en elbitset. A estebitset le sigue la codificación de los caracteres.

Salida por pantalla:

Introduzca uno de los siguientes comandos disponibles:

```
1. comprimir
2. descomprimir
3. salir
1
Escriba el path relativo de un fichero .txt a comprimir:
JuegosDePruebas/Prueba8/cansExtended.txt
Se inicia el proceso
El archivo /home/jan/Documents/PROP/PROPCompressor/EXE/LZSS/JuegosDePruebas/Prueba8/cansExtended.txt se ha comprimido correctamente!
Ha tardado 0.001263805s y se ha guardado en /home/jan/Documents/PROP/PROPCompressor/EXE/LZSS/JuegosDePruebas/Prueba8/cansExtended.lzss.
El cambio de tamaño pasa de 138B a 70B con diferencia de 68B que resulta en un 50.0% del archivo original.
```

Salida: cansExtended.lzss → vemos que es un archivo que contiene 4 bytes que codifican el tamaño delbitset, seguido delbitset y luego los caracteres codificados.

Veredicto: OK

Prueba 9

Descripción: Probar el método de compresión de la clase LZSS con un archivo de texto de gran tamaño que contiene una serie de caracteres ASCII y ASCIilexended.

Objetivo: Comprobar que comprime adecuadamente un fichero que contiene una serie de caracteres ASCII y ASCIilexended.

Driver Usado: LZSSDriver

Entrada: el_quijote.txt → fichero que contiene 1.060.259 bytes que son caracteres ASCII y ASCIilexended.

Resultado Esperado: res Esperado.lzss → El resultado esperado de la compresión es un archivo que contenga 4 bytes delbitset, en este caso elbitset va a continuación porque

como ya hay coincidencias estas se codifican y los flags están en el bitset. A este bitset le sigue la codificación de los caracteres.

Salida por pantalla:

Introduzca uno de los siguientes comandos disponibles:

```
1. comprimir
2. descomprimir
3. salir
1
Escriba el path relativo de un fichero .txt a comprimir:
JuegosDePruebas/Prueba9/el_quijote.txt
Se inicia el proceso
El archivo /home/jan/Documents/PROP/PROPCompressor/EXE/LZSS/JuegosDePruebas/Prueba9/el_quijote.txt se ha comprimido correctamente!
Ha tardado 4.236312779s y se ha guardado en /home/jan/Documents/PROP/PROPCompressor/EXE/LZSS/JuegosDePruebas/Prueba9/el_quijote.lzss.
El cambio de tamaño pasa de 10602598 a 511238B con diferencia de 549021B que resulta en un 48.0% del archivo original.
```

Salida: el_quijote.lzss → vemos que es un archivo que contiene 4 bytes que codifican el tamaño delbitset, seguido delbitset y luego los caracteres codificados.

Veredicto: OK

Prueba 10

Descripción: Probar el método de descompresión de la clase LZSS con un archivo de texto vacío comprimido en lzss .

Objetivo: Comprobar que descomprime adecuadamente un fichero vacío.

Driver Usado: LZSSDriver

Entrada: empty.lzss → Un archivo que contiene 4 bytes que almacenan el tamaño delbitset, que representan un 0. Por tanto 00 00 00 00.

Resultado Esperado: res Esperado.txt → un fichero de texto que no tiene ningún carácter.

Salida por pantalla:

Introduzca uno de los siguientes comandos disponibles:

```
1. comprimir
2. descomprimir
3. salir
2
Escriba el path relativo de un fichero .lzss a descomprimir:
JuegosDePruebas/Prueba10/empty.lzss
Se inicia el proceso
El archivo /home/jan/Documents/PROP/PROPCompressor/EXE/LZSS/JuegosDePruebas/Prueba10/empty.lzss se ha comprimido correctamente!
Ha tardado 3.67866E-4s y se ha guardado en /home/jan/Documents/PROP/PROPCompressor/EXE/LZSS/JuegosDePruebas/Prueba10/empty_out.txt.
El cambio de tamaño pasa de 4B a 0B con diferencia de -4B que resulta en un Infinity% del archivo original.
```

Salida: empty_out.txt → un fichero de texto sin ningún carácter.

Veredicto: OK

Prueba 11

Descripción: Probar el método de descompresión de la clase LZSS con un archivo lzss que contiene una serie de caracteres ASCII comprimidos.

Objetivo: Comprobar que descomprime adecuadamente un fichero que contiene una serie de caracteres ASCII.

Driver Usado: LZSSDriver

Entrada: abc.lzss → un archivo que contiene 4 bytes con valor hexa 00 00 00 00 que son la medida delbitset. seguido los bytes que codifican una serie de caracteres ASCII.

Resultado Esperado: res Esperado.txt → fichero que contiene 44 bytes que son caracteres ASCII.

Salida por pantalla:

Introduzca uno de los siguientes comandos disponibles:

1. comprimir
2. descomprimir
3. salir

2

Escriba el path relativo de un fichero .lzss a descomprimir:

JuegosDePruebas/Prueba1/abc.lzss

Se inicia el proceso

El archivo /home/jan/Documents/PROP/PROPCompressor/EXE/LZSS/JuegosDePruebas/Prueba1/abc.lzss se ha comprimido correctamente!

Ha tardado 2.73385E-4s y se ha guardado en /home/jan/Documents/PROP/PROPCompressor/EXE/LZSS/JuegosDePruebas/Prueba1/abc_out.txt.

El cambio de tamaño pasa de 48B a 44B con diferencia de -4B que resulta en un 109.0% del archivo original.

Salida: abc_out.txt → fichero que contiene 44 bytes adecuadamente descomprimidos.

Veredicto: OK

Prueba 12

Descripción: Probar el método de descompresión de la clase LZSS con un archivo de texto que contiene una serie de caracteres ASCII.

Objetivo: Comprobar que descomprime adecuadamente un fichero que contiene una serie de caracteres ASCII en el que sí que hay coincidencias probando por tanto que descomprime coincidencias .

Driver Usado: LZSSDriver

Entrada: cans.lzss → un archivo que contiene 4 bytes indicando el tamaño delbitset,seguidos delbitset y la codificación de los caracteres.

Resultado Esperado: res Esperado.txt → fichero que contiene 43 bytes que son caracteres ASCII.

Salida por pantalla:

Introduzca uno de los siguientes comandos disponibles:

1. comprimir
2. descomprimir
3. salir

2

Escriba el path relativo de un fichero .lzss a descomprimir:

JuegosDePruebas/Prueba12/cans.lzss

Se inicia el proceso

El archivo /home/jan/Documents/PROP/PROPCompressor/EXE/LZSS/JuegosDePruebas/Prueba12/cans.lzss se ha comprimido correctamente!

Ha tardado 2.58295E-4s y se ha guardado en /home/jan/Documents/PROP/PROPCompressor/EXE/LZSS/JuegosDePruebas/Prueba12/cans_out.txt.

El cambio de tamaño pasa de 35B a 43B con diferencia de 8B que resulta en un 81.0% del archivo original.

Salida: cans_out.txt → fichero que contiene 43 bytes adecuadamente descomprimidos.

Veredicto: OK

Prueba 13

Descripción: Probar el método de descompresión de la clase LZSS con un archivo de texto de gran tamaño que contiene una serie de caracteres ASCII y ASCIilexended comprimido con el algoritmo LZSS.

Objetivo: Comprobar que descomprime adecuadamente un fichero que contiene una serie de caracteres ASCII y ASCIilexended codificados.

Driver Usado: LZSSDriver

Entrada: el_quijote.lzss → un archivo que contiene 4 bytes que codifican el tamaño delbitset, seguido delbitset y luego los caracteres codificados.

Resultado Esperado: res Esperado.txt → 1.060.259bytes que son caracteres correctamente descomprimidos

Salida por pantalla:

Introduzca uno de los siguientes comandos disponibles:

1. comprimir
2. descomprimir
3. salir

2

Escriba el path relativo de un fichero .lzss a descomprimir:

JuegosDePruebas/Prueba13/el_quijote.lzss

Se inicia el proceso

El archivo /home/jan/Documents/PROP/PROPCompressor/EXE/LZSS/JuegosDePruebas/Prueba13/el_quijote.lzss se ha comprimido correctamente!
Ha tardado 0.123751954s y se ha guardado en /home/jan/Documents/PROP/PROPCompressor/EXE/LZSS/JuegosDePruebas/Prueba13/el_quijote_out.txt.
El cambio de tamaño pasa de 511238B a 10660259B con diferencia de 549021B que resulta en un 48.0% del archivo original.

Salida: el_quijote_out.txt → fichero que contiene 1.060.259 bytes que son caracteres ASCII y ASCIIextended adecuadamente descomprimidos.

Veredicto: OK

Prueba 14

Descripción: Comprobación de Garbage In → Garbage Out .

Objetivo: comprobar que si se intenta descomprimir un archivo que no ha sido comprimido con .lzss porque ha sido modificado o se ha creado a mano, el resultado es que no se realiza la descompresión.

Driver Usado: LZSSDriver

Entrada: Garbage.lzss → archivo generado a mano

Resultado Esperado: Una excepción y el programa produce un error.

Salida por pantalla:

Introduzca uno de los siguientes comandos disponibles:

1. comprimir
2. descomprimir
3. salir

2

Escriba el path relativo de un fichero .lzss a descomprimir:

JuegosDePruebas/Prueba14/Garbage.lzss

Se inicia el proceso

El archivo a descomprimir está corrupto o no se ha generado adecuadamente.

Salida: Devuelve una excepción de archivo corrupto.

Veredicto: OK

LZW

Prueba 1

Descripción: Probar el método de compresión de la clase LZW con un archivo de texto vacío.

Objetivo: Comprobar que comprime adecuadamente un fichero vacío.

Driver usado: LZWDriver

Entrada: empty.txt → un fichero de texto que no tiene ningún carácter.

Resultado esperado: res Esperado.lzw → El resultado esperado de la compresión es un archivo que no contenga bytes y de tamaño 0. Esta compresión no reduce el tamaño del archivo original, ya que no hay entrada de caracteres y se mantiene igual.

Salida por pantalla:

Introduzca uno de los siguientes comandos disponibles:

1. comprimir
2. descomprimir
3. salir

1

Escriba el path relativo de un fichero .txt a comprimir:

EXE/LZW/JuegosDePruebas/Prueba 1/empty.txt

Se inicia el proceso

El archivo C:\Users\sheida\Desktop\PROProject\PROPCompressor\EXE\LZW\JuegosDePruebas\Prueba 1\empty.txt se ha comprimido correctamente!
Ha tardado 0.0018408s y se ha guardado en C:\Users\sheida\Desktop\PROProject\PROPCompressor\EXE\LZW\JuegosDePruebas\Prueba 1\empty.lzw.

Salida: empty.lzw → vemos que es un archivo vacío de 0 bytes

Veredicto: OK

Prueba 2

Descripción: Probar el método de descompresión de la clase LZW con un archivo de texto vacío comprimido en lzw .

Objetivo: Comprobar que descomprime adecuadamente un fichero vacío.

Driver usado: LZWDriver

Entrada: empty.lzw → Un archivo que no contiene bytes.

Resultado esperado: res Esperado.txt → un fichero lzw que no tiene ningún byte y de tamaño 0.

Salida por pantalla:

Introduzca uno de los siguientes comandos disponibles:

1. comprimir
2. descomprimir
3. salir

2

Escriba el path relativo de un fichero .lzw a descomprimir:

EXE/LZW/JuegosDePruebas/Prueba 2/empty.lzw

Se inicia el proceso

El archivo C:\Users\sheida\Desktop\PROProject\PROPCompressor\EXE\LZW\JuegosDePruebas\Prueba 2\empty.lzw se ha descomprimido correctamente!
Ha tardado 0.0014932s y se ha guardado en C:\Users\sheida\Desktop\PROProject\PROPCompressor\EXE\LZW\JuegosDePruebas\Prueba 2\empty_out.txt

El cambio de tamaño pasa de 0B a 0B con diferencia de 0B que resulta en un 100.0% del archivo original.

Salida: empty_out.txt → un fichero de texto sin ningún carácter.

Veredicto: OK

Prueba 3

Descripción: Probar el método de compresión de la clase LZW con un archivo de texto que contiene una serie de caracteres ASCII.

Objetivo: Comprobar que comprime adecuadamente un fichero de texto que contiene una serie de caracteres ASCII.

Driver usado: LZWDriver

Entrada: abc.txt → fichero que contiene 39 bytes que son caracteres ASCII de letras y números.

Resultado esperado: res Esperado.lzw → El resultado esperado de la compresión es un archivo que contenga los bytes codificados con el algoritmo lzw del archivo original. El resultado no reduce el tamaño del archivo original porque no hay coincidencias.

Salida por pantalla:

Introduzca uno de los siguientes comandos disponibles:

1. comprimir
2. descomprimir
3. salir

1

Escriba el path relativo de un fichero .txt a comprimir:

EXE/LZW/JuegosDePruebas/Prueba 3/abc.txt

Se inicia el proceso

El archivo C:\Users\sheida\Desktop\PROProject\PROPCompressor\EXE\LZW\JuegosDePruebas\Prueba 3/abc.txt se ha comprimido correctamente!
Ha tardado 0.0288963s y se ha guardado en C:\Users\sheida\Desktop\PROProject\PROPCompressor\EXE\LZW\JuegosDePruebas\Prueba 3/abc.lzw.
El cambio de tamaño pasa de 39B a 152B con diferencia de -113B que resulta en un 389.0% del archivo original.

Salida: resultado.lzw → vemos que es un archivo que contiene conjuntos de 4 bytes codificando los bytes del archivo original.

Veredicto: OK

Prueba 4

Descripción: Probar el método de descompresión de la clase LZW con un archivo de texto que contiene una serie de caracteres ASCII.

Objetivo: Comprobar que descomprime adecuadamente un fichero de texto que contiene una serie de caracteres ASCII.

Driver usado: LZWDriver

Entrada: abc.lzw → fichero que contiene 152 bytes que son caracteres comprimidos.

Resultado esperado: res Esperado.txt → El resultado esperado de la descompresión es un archivo que contenga los bytes decodificados con el algoritmo lzw del archivo original.

Salida por pantalla:

Introduzca uno de los siguientes comandos disponibles:

1. comprimir
2. descomprimir
3. salir

2

Escriba el path relativo de un fichero .lzw a descomprimir:

EXE/LZW/JuegosDePruebas/Prueba 4/abc.lzw

Se inicia el proceso

El archivo C:\Users\sheida\Desktop\PROProject\PROPCompressor\EXE\LZW\JuegosDePruebas\Prueba 4/abc.lzw se ha descomprimido correctamente!
Ha tardado 0.0013365s y se ha guardado en C:\Users\sheida\Desktop\PROProject\PROPCompressor\EXE\LZW\JuegosDePruebas\Prueba 4/abc_out.txt
El cambio de tamaño pasa de 152B a 39B con diferencia de -113B que resulta en un 25.0% del archivo original.

Salida: abc_out.txt → vemos que es un archivo que contiene caracteres ASCII en bytes del archivo original.

Veredicto: OK

Prueba 5

Descripción: Probar el método de compresión de la clase LZW con un archivo de texto que contiene un carácter ASCIIextended.

Objetivo: Comprobar que comprime adecuadamente un fichero de texto que contiene un carácter ASCIIextended.

Driver usado: LZWDriver

Entrada: ASCIIextended.txt → fichero que contiene 2 bytes que forman una letra de ASCIIextended.

Resultado esperado: res Esperado.lzw → El resultado esperado de la compresión es un archivo que contenga los bytes codificados con el algoritmo lzw del archivo original. El resultado no reduce el tamaño del archivo original porque no hay repeticiones.

Salida por pantalla:

Introduzca uno de los siguientes comandos disponibles:

1. comprimir
2. descomprimir
3. salir

1

Escriba el path relativo de un fichero .txt a comprimir:

EXE/LZW/JuegosDePruebas/Prueba 5/ASCIIextended.txt

Se inicia el proceso

El archivo C:\Users\sheida\Desktop\PROProject\PROPCompressor\EXE\LZW\JuegosDePruebas\Prueba 5\ASCIIextended.txt se ha comprimido correctamente
Ha tardado 5.997E-4s y se ha guardado en C:\Users\sheida\Desktop\PROProject\PROPCompressor\EXE\LZW\JuegosDePruebas\Prueba 5\ASCIIextended.lzw.
El cambio de tamaño pasa de 28 a 8B con diferencia de -6B que resulta en un 400.0% del archivo original.

Salida: ASCIIextended.lzw → vemos que es un archivo que contiene conjuntos de 4 bytes codificando los bytes del archivo original.

Veredicto: OK

Prueba 6

Descripción: Probar el método de descompresión de la clase LZW con un archivo de texto que contiene un carácter ASCII extended.

Objetivo: Comprobar que descomprime adecuadamente un fichero de texto que contiene un carácter ASCII.

Driver usado: LZWDriver

Entrada: ASCIIextended.lzw → fichero que contiene 8 bytes que el carácter de ASCII extended codificado.

Resultado esperado: res Esperado.txt → El resultado esperado de la compresión es un archivo que contenga los bytes decodificados con el algoritmo lzw del archivo original. El resultado no reduce el tamaño del archivo original porque no hay repeticiones.

Salida por pantalla:

Introduzca uno de los siguientes comandos disponibles:

1. comprimir
2. descomprimir
3. salir

2

Escriba el path relativo de un fichero .lzw a descomprimir:

EXE/LZW/JuegosDePruebas/Prueba 6/ASCIIextended.lzw

Se inicia el proceso

El archivo C:\Users\sheida\Desktop\PROProject\PROPCompressor\EXE\LZW\JuegosDePruebas\Prueba 6\ASCIIextended.lzw se ha descomprimido correctamente!
Ha tardado 4.914E-4s y se ha guardado en C:\Users\sheida\Desktop\PROProject\PROPCompressor\EXE\LZW\JuegosDePruebas\Prueba 6\ASCIIextended_out.txt.
El cambio de tamaño pasa de 8B a 2B con diferencia de -6B que resulta en un 25.0% del archivo original.

Salida: ASCIIextended_out.txt → vemos que es un archivo que contiene el carácter ASCII extended del archivo original.

Veredicto: OK

Prueba 7

Descripción: Probar el método de compresión de la clase LZW con un archivo de texto que contiene varios caracteres ASCII extended.

Objetivo: Comprobar que comprime adecuadamente un fichero que contiene una serie de caracteres ASCII y ASCII Extended que por naturaleza se codifica en dos bytes y no en uno.

Driver usado: LZWDriver

Entrada: abcASCIIextended.txt → Fichero de texto de 73 bytes.

Resultado esperado: res Esperado.lzw → El resultado esperado de la compresión es un archivo que contenga los bytes codificados con el algoritmo lzw del archivo original. El resultado no reduce el tamaño del archivo original porque no hay coincidencias.

Salida por pantalla:

Introduzca uno de los siguientes comandos disponibles:

1. comprimir
2. descomprimir
3. salir

1

Escriba el path relativo de un fichero .txt a comprimir:

EXE\LZW\JuegosDePruebas\Prueba 7\abcASCIIextended.txt

Se inicia el proceso

El archivo C:\Users\sheida\Desktop\PROProject\PROPCompressor\EXE\LZW\JuegosDePruebas\Prueba 7\abcASCIIextended.txt se ha comprimido correctamente!
Ha tardado 0.0016439s y se ha guardado en C:\Users\sheida\Desktop\PROProject\PROPCompressor\EXE\LZW\JuegosDePruebas\Prueba 7\abcASCIIextended.lzw.
El cambio de tamaño pasa de 73B a 252B con diferencia de -179B que resulta en un 345.0% del archivo original.

Salida: abcASCIIextended.lzw → vemos que es un archivo que contiene conjuntos de 4 bytes codificando los bytes del archivo original.

Veredicto: OK

Prueba 8

Descripción: Probar el método de descompresión de la clase LZW con un archivo de texto que contiene caracteres ASCII extended.

Objetivo: Comprobar que descomprime adecuadamente un fichero comprimido con lzw que contenía una serie de caracteres ASCII y ASCII Extended que por naturaleza se codifica en dos bytes y no en uno.

Driver usado: LZWDriver

Entrada: abcASCIIextended.lzw

Resultado esperado: res Esperado.txt → El resultado esperado de la descompresión es un archivo que contenga los bytes decodificados con el algoritmo lzw del archivo original. El resultado no reduce el tamaño del archivo original porque no hay repeticiones.

Salida por pantalla:

Introduzca uno de los siguientes comandos disponibles:

1. comprimir
2. descomprimir
3. salir

2

Escriba el path relativo de un fichero .lzw a descomprimir:

EXE/LZW/JuegosDePruebas/Prueba 8/abcASCIIfextended.lzw

Se inicia el proceso

El archivo C:\Users\sheida\Desktop\PROProject\PROPCompressor\EXE\LZW\JuegosDePruebas\Prueba 8/abcASCIIfextended.lzw se ha descomprimido correctamente!

Ha tardado 0.0022259s y se ha guardado en C:\Users\sheida\Desktop\PROProject\PROPCompressor\EXE\LZW\JuegosDePruebas\Prueba 8/abcASCIIfextended_out.txt.

El cambio de tamaño pasa de 232B a 66B con diferencia de -166B que resulta en un 28.0% del archivo original.

Salida: abcASCIIfextended_out.txt → vemos que es un archivo que contiene los bytes del archivo original.

Veredicto: OK

Prueba 9

Descripción: Probar el método de compresión de la clase LZW con un archivo de texto con coincidencias de caracteres ASCII y ASCII extended.

Objetivo: Comprobar que comprime adecuadamente un fichero que contiene una serie de caracteres.

Driver usado: LZWDriver

Entrada: coco.txt → El fichero de texto contiene 622 bytes.

Resultado esperado: res Esperado.lzw → El resultado esperado de la compresión es un archivo que contenga los bytes codificados con el algoritmo lzw del archivo original. El resultado no reduce el tamaño del archivo original porque no hay suficientes coincidencias.

Salida por pantalla:

Introduzca uno de los siguientes comandos disponibles:

1. comprimir
2. descomprimir
3. salir

1

Escriba el path relativo de un fichero .txt a comprimir:

EXE/LZW/JuegosDePruebas/Prueba 9/coco.txt

Se inicia el proceso

El archivo C:\Users\sheida\Desktop\PROProject\PROPCompressor\EXE\LZW\JuegosDePruebas\Prueba 9/coco.txt se ha comprimido correctamente!

Ha tardado 0.0024055s y se ha guardado en C:\Users\sheida\Desktop\PROProject\PROPCompressor\EXE\LZW\JuegosDePruebas\Prueba 9/coco.lzw.

El cambio de tamaño pasa de 119B a 204B con diferencia de -85B que resulta en un 171.0% del archivo original.

Salida: coco.lzw → vemos que es un archivo que contiene conjuntos de 4 bytes codificando los bytes del archivo original.

Veredicto: OK

Prueba 10

Descripción: Probar el método de descompresión de la clase LZW con un archivo de texto que contiene caracteres ASCII y ASCII extended.

Objetivo: Comprobar que descomprime adecuadamente un fichero comprimido con lzw que contenía una serie de caracteres con algunas repeticiones de caracteres y conjuntos.

Driver usado: LZWDriver

Entrada: coco.lzw

Resultado esperado: res Esperado.txt → El resultado esperado de la descompresión es un archivo que contenga los bytes decodificados con el algoritmo lzw del archivo original.

Salida por pantalla:

Introduzca uno de los siguientes comandos disponibles:

1. comprimir
2. descomprimir
3. salir

2

Escriba el path relativo de un fichero .lzw a descomprimir:

EXE/LZW/JuegosDePruebas/Prueba 10/coco.lzw

Se inicia el proceso

El archivo C:\Users\sheida\Desktop\PROProject\PROPCompressor\EXE\LZW\JuegosDePruebas\Prueba 10\coco.lzw se ha descomprimido correctamente!

Ha tardado 0.0035644s y se ha guardado en C:\Users\sheida\Desktop\PROProject\PROPCompressor\EXE\LZW\JuegosDePruebas\Prueba 10\coco_out.txt

El cambio de tamaño pasa de 616B a 622B con diferencia de 6B que resulta en un 100.0% del archivo original.

Salida: coco_out.txt → vemos que es un archivo que contiene una frase repetida varias veces del fichero de texto orginal.

Veredicto: OK

Prueba 11

Descripción: Probar el método de compresión de la clase LZW con un archivo de texto de un libro extenso de 1.062.445 bytes.

Objetivo: Comprobar que comprime adecuadamente un fichero que contiene un libro de texto.

Driver usado: LZWDriver

Entrada: el_quijote.txt

Resultado esperado: res Esperado.lzw → El resultado esperado de la compresión es un archivo que contenga los bytes codificados con el algoritmo lzw del archivo original. El resultado reduce el tamaño del archivo original.

Salida por pantalla:

Introduzca uno de los siguientes comandos disponibles:

1. comprimir
2. descomprimir
3. salir

1

Escriba el path relativo de un fichero .txt a comprimir:

EXE/LZW/JuegosDePruebas/Prueba 11/el_quijote.txt

Se inicia el proceso

El archivo C:\Users\sheida\Desktop\PROProject\PROPCompressor\EXE\LZW\JuegosDePruebas\Prueba 11\el_quijote.txt se ha comprimido correctamente!

Ha tardado 1.367217s y se ha guardado en C:\Users\sheida\Desktop\PROProject\PROPCompressor\EXE\LZW\JuegosDePruebas\Prueba 11\el_quijote.lzw.

El cambio de tamaño pasa de 1062445B a 730576B con diferencia de 331869B que resulta en un 68.0% del archivo original.

Salida: el_quijote.lzw → vemos que es un archivo que contiene conjuntos de 4 bytes para codificar los bytes del archivo original.

Veredicto: OK

Prueba 12

Descripción: Probar el método de descompresión de la clase LZW con un archivo de texto que contiene 733.184 bytes.

Objetivo: Comprobar que descomprime adecuadamente un fichero comprimido con lzw que contenía una serie de caracteres ASCII Extended con algunas repeticiones de caracteres y conjuntos.

Driver usado: LZWDriver

Entrada: el_quijote.lzw

Resultado esperado: res Esperado.txt → El resultado esperado de la descompresión es un archivo que contenga los bytes decodificados con el algoritmo lzw del archivo original. El resultado no reduce el tamaño del archivo original porque no suficientes repeticiones.

Salida por pantalla:

Introduzca uno de los siguientes comandos disponibles:

1. comprimir
2. descomprimir
3. salir

2

Escriba el path relativo de un fichero .lzw a descomprimir:

EXE/LZW/JuegosDePruebas/Prueba 12/el_quijote.lzw

Se inicia el proceso

El archivo C:\Users\sheida\Desktop\PROProject\PROPCompressor\EXE\LZW\JuegosDePruebas\Prueba 12\el_quijote.lzw se ha descomprimido correctamente!
Ha tardado 0.6494497s y se ha guardado en C:\Users\sheida\Desktop\PROProject\PROPCompressor\EXE\LZW\JuegosDePruebas\Prueba 12\el_quijote_out.txt.
El cambio de tamaño pasa de 730576B a 1062445B con diferencia de 331869B que resulta en un 145.0% del archivo original.

Salida: el_quijote_out.txt → vemos que es un archivo que contiene el libro extenso orginal.

Veredicto: OK

Prueba 13

Descripción: Probar el método de descompresión de la clase LZW con Garbage In → Garbage Out.

Objetivo: Comprobar que si se intenta descomprimir un archivo que no ha sido comprimido con .lzw porque ha sido modificado o se ha creado a mano, el resultado es que no se realiza la descompresión.

Driver usado: LZWDriver

Entrada: Garbage.lzw

Resultado esperado: Devuelve una excepción y ningún fichero creado.

Salida por pantalla:

Introduzca uno de los siguientes comandos disponibles:

1. comprimir
2. descomprimir
3. salir

2

Escriba el path relativo de un fichero .lzw a descomprimir:

EXE/LZW/JuegosDePruebas/Prueba 13/Garbage.lzw

Se inicia el proceso

El archivo a descomprimir está corrupto o no se ha generado adecuadamente.

Salida: No se ha realizado la compresión y devuelve una excepción. No se ha creado ningún fichero porque no se ha cumplido la precondition.

Veredicto: OK

LZ78

Prueba 1

Descripción: Probar el método de compresión de la clase LZ78 con un archivo de texto vacío.

Objetivo: Comprobar que comprime adecuadamente un fichero vacío.

Driver Usado: LZ78Driver.

Entrada: empty.txt → un fichero de texto que no tiene ningún carácter.

Resultado Esperado: res Esperado.lz78 → El resultado esperado de la compresión es un archivo que no contenga ningún dato, igual que el original.

Salida por pantalla:

```
Introduzca uno de los siguientes comandos disponibles:  
1. comprimir  
2. descomprimir  
3. salir  
1  
Escriba el path relativo de un fichero .txt a comprimir:  
JuegosDePruebas/Prueba1/empty.txt  
Se inicia el proceso  
El archivo /home/javidran/Documents/GitKraken/PROPCompressor/EXE/LZ78/JuegosDePruebas/Prueba1/empty.txt se ha comprimido correctamente!  
Ha tardado 7.12735E-4s y se ha guardado en /home/javidran/Documents/GitKraken/PROPCompressor/EXE/LZ78/JuegosDePruebas/Prueba1/empty.lz78.  
El cambio de tamaño pasa de 0B a 0B con diferencia de 0B que resulta en un 100.0% del archivo original.
```

Salida: empty.lz78 → No contiene ningún byte, como esperado.

Veredicto: OK

Prueba 2

Descripción: Probar el método de compresión de la clase LZ78 con un archivo de texto que contiene un solo carácter ASCII.

Objetivo: Comprobar que comprime adecuadamente un fichero que contiene un solo carácter ASCII.

Driver Usado: LZ78Driver

Entrada: a.txt

Resultado Esperado: res Esperado.lz78 → El resultado esperado de la compresión es que se comprimirá correctamente el archivo aunque el tamaño del nuevo archivo incrementará, ya que el contenido del original es demasiado pequeño como para que compense el uso del compresor.

Salida por pantalla:

```
Escriba el path relativo de un fichero .txt a comprimir:  
JuegosDePruebas/Prueba2/a.txt  
Se inicia el proceso  
El archivo /home/javidran/Documents/GitKraken/PROPCompressor/EXE/LZ78/JuegosDePruebas/Prueba2/a.txt se ha comprimido correctamente!  
Ha tardado 0.001627084s y se ha guardado en /home/javidran/Documents/GitKraken/PROPCompressor/EXE/LZ78/JuegosDePruebas/Prueba2/a.lz78.  
El cambio de tamaño pasa de 2B a 4B con diferencia de -2B que resulta en un 200.0% del archivo original.
```

Salida: a.lz78 -> Vemos como el archivo se ha comprimido correctamente y pasa a ocupar más tamaño que el original debido al pequeño tamaño del mismo.

Veredicto: OK

Prueba 3

Descripción: Probar el método de compresión de la clase LZ78 con un archivo de texto que contiene un solo carácter ASCII Extended.

Objetivo: Comprobar que comprime adecuadamente un fichero que contiene un solo carácter ASCII Extended que por naturaleza se codifica en dos bytes y no en uno.

Driver Usado: LZ78Driver

Entrada: ASCIilexended.txt

Resultado Esperado: res Esperado.lz78 → El resultado esperado de la compresión es que se comprimirá correctamente el archivo aunque el tamaño del nuevo archivo incrementará, ya que el contenido del original es demasiado pequeño como para que compense el uso del compresor.

Salida por pantalla:

```
Introduzca uno de los siguientes comandos disponibles:  
1. comprimir  
2. descomprimir  
3. salir  
1  
Escriba el path relativo de un fichero .txt a comprimir:  
JuegosDePruebas/Prueba3/ASCIilexended.txt  
Se inicia el proceso  
El archivo /home/javidran/Documents/GitKraken/PROPCompressor/EXE/LZ78/JuegosDePruebas/Prueba3/ASCIilexended.txt se ha comprimido correctamente!  
Ha tardado 0.001675906s y se ha guardado en /home/javidran/Documents/GitKraken/PROPCompressor/EXE/LZ78/JuegosDePruebas/Prueba3/ASCIilexended.lz78.  
El cambio de tamaño pasa de 3B a 6B con diferencia de -3B que resulta en un 200.0% del archivo original.
```

Salida: ASCIilexended.lz78 → Vemos como el archivo se ha comprimido correctamente y pasa a ocupar más tamaño que el original debido al pequeño tamaño del mismo.

Veredicto: OK

Prueba 4

Descripción: Probar el método de compresión de la clase LZ78 con un archivo de texto que contiene una serie de caracteres ASCII.

Objetivo: Comprobar que comprime adecuadamente un fichero que contiene una serie de caracteres ASCII.

Driver Usado: LZ78Driver

Entrada: abc.txt → fichero que contiene 44 bytes que son caracteres ASCII.

Resultado Esperado: res Esperado.lz78 → El resultado esperado de la compresión es que la compresión se realice adecuadamente, aunque hay posibilidad de que el fichero ocupe más ya que el formato de la compresión no resulta beneficiosa para un archivo tan pequeño.

Salida por pantalla:

```
Introduzca uno de los siguientes comandos disponibles:  
1. comprimir  
2. descomprimir  
3. salir  
1  
Escriba el path relativo de un fichero .txt a comprimir:  
JuegosDePruebas/Prueba4/abc.txt  
Se inicia el proceso  
El archivo /home/javidran/Documents/GitKraken/PROPCompressor/EXE/LZ78/JuegosDePruebas/Prueba4/abc.txt se ha comprimido correctamente!  
Ha tardado 0.005651092s y se ha guardado en /home/javidran/Documents/GitKraken/PROPCompressor/EXE/LZ78/JuegosDePruebas/Prueba4/abc.lz78.  
El cambio de tamaño pasa de 44B a 78B con diferencia de -34B que resulta en un 177.0% del archivo original.
```

Salida: abc.lz78 → Vemos como el archivo se ha comprimido correctamente y pasa a ocupar más tamaño que el original debido al pequeño tamaño del mismo.

Veredicto: OK

Prueba 5

Descripción: Probar el método de compresión de la clase LZ78 con un archivo de texto de pequeño tamaño que contiene una serie combinada de caracteres ASCII y ASCII Extended.

Objetivo: Comprobar que comprime adecuadamente un fichero que contiene una serie combinada de caracteres ASCII y ASCII Extended.

Driver Usado: LZ78Driver

Entrada: cans.txt

Resultado Esperado: res Esperado.lz78 → El resultado esperado de la compresión es que la compresión se realice adecuadamente.

Salida por pantalla:

```
Introduzca uno de los siguientes comandos disponibles:  
1. comprimir  
2. descomprimir  
3. salir  
1  
Escriba el path relativo de un fichero .txt a comprimir:  
JuegosDePruebas/Prueba5/cans.txt  
Se inicia el proceso  
El archivo /home/javidran/Documents/GitKraken/PROPCompressor/EXE/LZ78/JuegosDePruebas/Prueba5/cans.txt se ha comprimido correctamente!  
Ha tardado 4.24397E-4s y se ha guardado en /home/javidran/Documents/GitKraken/PROPCompressor/EXE/LZ78/JuegosDePruebas/Prueba5/cans.lz78.  
El cambio de tamaño pasa de 67B a 66B con diferencia de 1B que resulta en un 98.0% del archivo original.
```

Salida: cans.lz78 → vemos que es un archivo que contiene la estructuración correcta de un archivo comprimido y tiene prácticamente una estructura adecuada para poder representar cierta compresión respecto al archivo original.

Veredicto: OK

Prueba 6

Descripción: Probar el método de compresión de la clase LZ78 con un archivo de texto de gran tamaño que contiene una serie de caracteres ASCII y ASCIilexended.

Objetivo: Comprobar que comprime adecuadamente un fichero que contiene una serie de caracteres ASCII y ASCIilexended.

Driver Usado: LZ78Driver

Entrada: el_quijote.txt → fichero que contiene 1.060.259 bytes que son caracteres ASCII y ASCIilexended.

Resultado Esperado: res Esperado.lz78 → El resultado esperado es un archivo con la estructura adecuada de compresión LZ78 y una drástica reducción del tamaño del archivo.

Salida por pantalla:

```
Introduzca uno de los siguientes comandos disponibles:  
1. comprimir  
2. descomprimir  
3. salir  
1  
Escriba el path relativo de un fichero .txt a comprimir:  
JuegosDePruebas/Prueba6/el_quijote.txt  
Se inicia el proceso  
El archivo /home/javidran/Documents/GitKraken/PROPCompressor/EXE/LZ78/JuegosDePruebas/Prueba6/el_quijote.txt se ha comprimido correctamente!  
Ha tardado 0.334680936s y se ha guardado en /home/javidran/Documents/GitKraken/PROPCompressor/EXE/LZ78/JuegosDePruebas/Prueba6/el_quijote.lz78.  
El cambio de tamaño pasa de 1060259B a 571982B con diferencia de 488277B que resulta en un 53.0% del archivo original.
```

Salida: el_quijote_out.txt → vemos como efectivamente el tamaño del archivo se ha visto reducido.

Veredicto: OK

Prueba 7

Descripción: Probar el método de descompresión de la clase LZ78 con un archivo de texto vacío comprimido en LZ78.

Objetivo: Comprobar que descomprime adecuadamente un fichero vacío.

Driver Usado: LZ78Driver

Entrada: empty.lz78 → Archivo comprimido generado al comprimir en la Prueba 1.

Resultado Esperado: res Esperado.txt → un fichero de texto que no tiene ningún carácter.

Salida por pantalla:

```
Introduzca uno de los siguientes comandos disponibles:  
1. comprimir  
2. descomprimir  
3. salir  
2  
Escriba el path relativo de un fichero .lz78 a descomprimir:  
JuegosDePruebas/Prueba7/empty.lz78  
Se inicia el proceso  
El archivo /home/javidran/Documents/GitKraken/PROPCompressor/EXE/LZ78/JuegosDePruebas/Prueba7/empty.lz78 se ha descomprimido correctamente!  
Ha tardado 2.1376E-5s y se ha guardado en /home/javidran/Documents/GitKraken/PROPCompressor/EXE/LZ78/JuegosDePruebas/Prueba7/empty_out.txt.  
El cambio de tamaño pasa de 0B a 0B con diferencia de 0B que resulta en un 100.0% del archivo original.
```

Salida: empty_out.txt → un fichero de texto sin ningún carácter.

Veredicto: OK

Prueba 8

Descripción: Probar el método de descompresión de la clase LZ78 con un archivo de texto con solo un carácter comprimido en LZ78.

Objetivo: Comprobar que descomprime adecuadamente un fichero con 1 carácter.

Driver Usado: LZ78Driver

Entrada: a.lz78 → Archivo comprimido generado al comprimir en la Prueba 2.

Resultado Esperado: res Esperado.txt → un fichero de texto que contendrá un solo carácter.

Salida por pantalla:

```
Introduzca uno de los siguientes comandos disponibles:  
1. comprimir  
2. descomprimir  
3. salir  
2  
Escriba el path relativo de un fichero .lz78 a descomprimir:  
JuegosDePruebas/Prueba8/a.lz78  
Se inicia el proceso  
El archivo /home/javidran/Documents/GitKraken/PROPCompressor/EXE/LZ78/JuegosDePruebas/Prueba8/a.lz78 se ha descomprimido correctamente!  
Ha tardado 0.00303699s y se ha guardado en /home/javidran/Documents/GitKraken/PROPCompressor/EXE/LZ78/JuegosDePruebas/Prueba8/a_out.txt.  
El cambio de tamaño pasa de 4B a 2B con diferencia de -2B que resulta en un 50.0% del archivo original.
```

Salida: a_out.txt → un fichero de texto con un carácter.

Veredicto: OK

Prueba 9

Descripción: Probar el método de descompresión de la clase LZ78 con un archivo de texto con solo un carácter de ASCIIextended comprimido en LZ78.

Objetivo: Comprobar que descomprime adecuadamente un fichero con 1 carácter ASCIIextended.

Driver Usado: LZ78Driver

Entrada: ASCIIextended.lz78 → Archivo comprimido generado al comprimir en la Prueba 3.

Resultado Esperado: res Esperado.txt → un fichero de texto que contendrá un solo carácter.

Salida por pantalla:

```
Introduzca uno de los siguientes comandos disponibles:  
1. comprimir  
2. descomprimir  
3. salir  
2  
Escriba el path relativo de un fichero .lz78 a descomprimir:  
JuegosDePruebas/Prueba9/ASCIIextended.lz78  
Se inicia el proceso  
El archivo /home/javidran/Documents/GitKraken/PROPCompressor/EXE/LZ78/JuegosDePruebas/Prueba9/ASCIIextended.lz78 se ha descomprimido correctamente!  
Ha tardado 6.8288E-5s y se ha guardado en /home/javidran/Documents/GitKraken/PROPCompressor/EXE/LZ78/JuegosDePruebas/Prueba9/ASCIIextended_out.txt.  
El cambio de tamaño pasa de 6B a 3B con diferencia de -3B que resulta en un 50.0% del archivo original.
```

Salida: ASCIIextended_out.txt → un fichero de texto con un carácter ASCIIextended.

Veredicto: OK

Prueba 10

Descripción: Probar el método de descompresión de la clase LZ78 con un archivo de texto de pequeño tamaño que contiene una serie combinada de caracteres ASCII y ASCII Extended comprimido en LZ78.

Objetivo: Comprobar que descomprime adecuadamente un fichero con cualquier tipo de carácter.

Driver Usado: LZ78Driver

Entrada: abc.lz78 → Archivo comprimido generado al comprimir en la Prueba 4.

Resultado Esperado: res Esperado.txt → un fichero de texto exactamente igual al original antes de comprimirlo de la Prueba 4.

Salida por pantalla:

```
Introduzca uno de los siguientes comandos disponibles:  
1. comprimir  
2. descomprimir  
3. salir  
2  
Escriba el path relativo de un fichero .lz78 a descomprimir:  
JuegosDePruebas/Prueba10/abc.lz78  
Se inicia el proceso  
El archivo /home/javidran/Documents/GitKraken/PROPCompressor/EXE/LZ78/JuegosDePruebas/Prueba10/abc.lz78 se ha descomprimido correctamente!  
Ha tardado 7.1366E-4s y se ha guardado en /home/javidran/Documents/GitKraken/PROPCompressor/EXE/LZ78/JuegosDePruebas/Prueba10/abc_out.txt.  
El cambio de tamaño pasa de 78B a 44B con diferencia de -34B que resulta en un 56.0% del archivo original.
```

Salida: abc_out.txt → un fichero de texto igual al original sin comprimir de la Prueba 4.

Veredicto: OK

Prueba 11

Descripción: Probar el método de descompresión de la clase LZ78 con un archivo de texto de pequeño tamaño que contiene una serie combinada de caracteres ASCII y ASCII Extended comprimido en LZ78.

Objetivo: Comprobar que descomprime adecuadamente un fichero con cualquier tipo de carácter.

Driver Usado: LZ78Driver

Entrada: cans.lz78 → Archivo comprimido generado al comprimir en la Prueba 5.

Resultado Esperado: res Esperado.txt → un fichero de texto exactamente igual al original antes de comprimirlo de la Prueba 5.

Salida por pantalla:

```
Introduzca uno de los siguientes comandos disponibles:  
1. comprimir  
2. descomprimir  
3. salir  
2  
Escriba el path relativo de un fichero .lz78 a descomprimir:  
JuegosDePruebas/Prueba11/cans.lz78  
Se inicia el proceso  
El archivo /home/javidran/Documents/GitKraken/PROPCompressor/EXE/LZ78/JuegosDePruebas/Prueba11/cans.lz78 se ha descomprimido correctamente!  
Ha tardado 3.85963E-4s y se ha guardado en /home/javidran/Documents/GitKraken/PROPCompressor/EXE/LZ78/JuegosDePruebas/Prueba11/cans_out.txt.  
El cambio de tamaño pasa de 66B a 67B con diferencia de 1B que resulta en un 101.0% del archivo original.
```

Salida: cans_out.txt → un fichero de texto igual al original sin comprimir de la Prueba 5.

Veredicto: OK

Prueba 12

Descripción: Probar el método de descompresión de la clase LZ78 con un archivo de texto de gran tamaño que contiene una serie combinada de caracteres ASCII y ASCII Extended comprimido en LZ78.

Objetivo: Comprobar que descomprime adecuadamente un fichero con cualquier tipo de carácter.

Driver Usado: LZ78Driver

Entrada: el_quijote.lz78 → Archivo comprimido generado al comprimir en la Prueba 5.

Resultado Esperado: res Esperado.txt → un fichero de texto exactamente igual al original antes de comprimirlo de la Prueba 6.

Salida por pantalla:

```
Introduzca uno de los siguientes comandos disponibles:  
1. comprimir  
2. descomprimir  
3. salir  
2  
Escriba el path relativo de un fichero .lz78 a descomprimir:  
JuegosDePruebas/Prueba12/el_quijote.lz78  
Se inicia el proceso  
El archivo /home/javidran/Documents/GitKraken/PROPCompressor/EXE/LZ78/JuegosDePruebas/Prueba12/el_quijote.lz78 se ha descomprimido correctamente!  
Ha tardado 0.212056087s y se ha guardado en /home/javidran/Documents/GitKraken/PROPCompressor/EXE/LZ78/JuegosDePruebas/Prueba12/el_quijote_out.txt.  
El cambio de tamaño pasa de 571982B a 1060259B con diferencia de 488277B que resulta en un 185.0% del archivo original.
```

Salida: el_quijote_out.txt → un fichero de texto igual al original sin comprimir de la Prueba 6.

Veredicto: OK

Prueba 13

Descripción: Comprobación de Garbage In → Garbage Out .

Objetivo: Comprobar que si se intenta descomprimir un archivo que no ha sido comprimido con .lz78 porque ha sido modificado o se ha creado a mano, el resultado es que no se realiza la descompresión.

Driver Usado: LZ78Driver

Entrada: Garbage.lz78 → archivo generado a mano

Resultado Esperado: que no se genere ningún resultado y el programa produce garbage.

Salida por pantalla:

```
Introduzca uno de los siguientes comandos disponibles:  
1. comprimir  
2. descomprimir  
3. salir  
2  
Escriba el path relativo de un fichero .lz78 a descomprimir:  
EXE/LZ78/JuegosDePruebas/Prueba 13/Garbage.lz78  
Se inicia el proceso  
El archivo a descomprimir está corrupto o no se ha generado adecuadamente.
```

Salida: efectivamente no se ha generado ningún resultado y el programa ha producido garbage.

Veredicto: OK

JPEG

Prueba 1

Descripción: Probar el método de compresión de la clase JPEG con un archivo de imagen ppm vacío.

Objetivo: Comprobar que se produce y trata una excepción de formato erróneo.

Driver Usado: JPEGDriver

Entrada: void.ppm → un fichero de imagen ppm que no tiene ningún byte de información. También se espera una calidad de compresión, del 1 al 7, siendo 1 la más baja posible y 7 la más alta.

Resultado Esperado: Activación y tratado de una excepción de formato erróneo.

Salida por pantalla:

```
Bienvenido al driver para el algoritmo de JPEG
```

```
Introduzca uno de los siguientes comandos disponibles:
```

1. comprimir
2. descomprimir
3. salir

```
1
```

```
Escriba el path relativo de un fichero .ppm a comprimir:
```

```
EXE/JocsDeProves/JPEG/Prueba 1/void.ppm
```

```
Indique la calidad de compresión a usar (del 1 al 7)
```

```
4
```

```
Se inicia el proceso
```

```
El formato de .ppm no es correcto!
```

Salida:

Veredicto: OK

Prueba 2

Descripción: Probar el método de compresión de la clase JPEG con un archivo de imagen con formato erróneo tanto en el header como en el pixelmap.

Objetivo: Comprobar que se produce y trata una excepción de formato erróneo.

Driver Usado: JPEGDriver

Entrada: wrong.ppm (o cualquier otro fichero wrong de los provistos) → un fichero de imagen ppm que tiene errores de header y inconsistencia entre información del pixel map y las dimensiones supuestas de la imagen. También se espera una calidad de compresión, del 1 al 7, siendo 1 la más baja posible y 7 la más alta.

Resultado Esperado: Activación y tratado de una excepción de formato erróneo.

Salida por pantalla:

Introduzca uno de los siguientes comandos disponibles:

```
1. comprimir
2. descomprimir
3. salir
1
Escriba el path relativo de un fichero .ppm a comprimir:
EXE/JocsDeProves/JPEG/Prueba 2/wrong.ppm
Indique la calidad de compresión a usar (del 1 al 7)
7
Se inicia el proceso
El formato de .ppm no es correcto!
```

Salida:

Veredicto: OK

Prueba 3

Descripción: Probar el método de compresión de la clase JPEG con un archivo de imagen con formato correcto.

Objetivo: Comprobar que se comprime la imagen correctamente.

Driver Usado: JPEGDriver

Entrada: fall_2.ppm (se puede usar cualquiera de las otras imágenes provistas) → un fichero de imagen ppm con magicNumber P6, dimensiones positivas y rgbMaxVal con valor 255. También se espera una calidad de compresión, del 1 al 7, siendo 1 la más baja posible y 7 la más alta (se usa 4, pero se podría usar cualquier otro valor dentro del rango a voluntad).

Resultado Esperado: res Esperado.imgc → Compresión de imagen ppm.

Salida por pantalla:

Introduzca uno de los siguientes comandos disponibles:

```
1. comprimir
2. descomprimir
3. salir
1
Escriba el path relativo de un fichero .ppm a comprimir:
EXE/JocsDeProves/JPEG/Prueba 2/wrong.ppm
Indique la calidad de compresión a usar (del 1 al 7)
7
Se inicia el proceso
El formato de .ppm no es correcto!
Introduzca uno de los siguientes comandos disponibles:

1. comprimir
2. descomprimir
3. salir
1
Escriba el path relativo de un fichero .ppm a comprimir:
EXE/JocsDeProves/JPEG/Prueba 3/falls_2.ppm
Indique la calidad de compresión a usar (del 1 al 7)
4
Se inicia el proceso
El archivo D:\Documents\GitHub\PROPCompressorHome\PROPCompressor\EXE\JocsDeProves\JPEG\Prueba 3\falls_2.ppm se ha comprimido correctamente!
Ha tardado 3.9380309s y se ha guardado en D:\Documents\GitHub\PROPCompressorHome\PROPCompressor\EXE\JocsDeProves\JPEG\Prueba 3\falls_2.imgc.
El cambio de tamaño pasa de 2332861B a 1169298B con diferencia de 1163563B que resulta en un 50.0% del archivo original.
```

Salida: res Esperado.imgc → Imagen ppm descomprimida..

Veredicto: OK

Prueba 4

Descripción: Probar el método de descompresión de la clase JPEG con un archivo de imagen imgc vacío.

Objetivo: Comprobar que se produce y trata una excepción de formato erróneo.

Driver Usado: JPEGDriver

Entrada: void.imgc → un fichero de imagen ppm que no tiene ningún byte de información.

Resultado Esperado: Activación y tratado de una excepción de formato erróneo.

Salida por pantalla:

```
Bienvenido al driver para el algoritmo de JPEG
```

```
Introduzca uno de los siguientes comandos disponibles:
```

- 1. comprimir
- 2. descomprimir
- 3. salir

```
2
```

```
Escriba el path relativo de un fichero .imgc a descomprimir:
```

```
EXE/JocsDeProves/JPEG/Prueba 4/void.imgc
```

```
Se inicia el proceso
```

```
El formato de .imgc no es correcto!
```

Salida:

Veredicto: OK

Prueba 5

Descripción: Probar el método de descompresión de la clase JPEG con un archivo de imagen con formato erróneo tanto en el header como en el pixelmap.

Objetivo: Comprobar que se produce y trata una excepción de formato erróneo.

Driver Usado: JPEGDriver

Entrada: wrong.imgc (o cualquier otro fichero wrong de los provistos) → un fichero de imagen ppm que tiene errores de header y inconsistencia entre información del pixel map y las dimensiones supuestas de la imagen.

Resultado Esperado: Activación y tratado de una excepción de formato erróneo.

Salida por pantalla:

```
Introduzca uno de los siguientes comandos disponibles:
```

- 1. comprimir
- 2. descomprimir
- 3. salir

```
2
```

```
Escriba el path relativo de un fichero .imgc a descomprimir:
```

```
EXE/JocsDeProves/JPEG/Prueba 5/wrong.imgc
```

```
Se inicia el proceso
```

```
El formato de .imgc no es correcto!
```

Salida:

Veredicto: OK

Prueba 6

Descripción: Probar el método de descompresión de la clase JPEG con un archivo de imagen con formato correcto.

Objetivo: Comprobar que se comprime la imagen correctamente.

Driver Usado: JPEGDriver

Entrada: falls_2.imgc (o cualquier otro de los provistos) → un fichero de imagen ppm con magicNumber P6, dimensiones positivas, rgbMaxVal con valor 255 y calidad con valor del 1 al 7.

Resultado Esperado: res Esperado.ppm → Imagen ppm descomprimida.

Salida por pantalla:

Introduzca uno de los siguientes comandos disponibles:

```
1. comprimir
2. descomprimir
3. salir
2
Escriba el path relativo de un fichero .imgc a descomprimir:
EXE/JPEG/JuegosDePruebas/Prueba6/falls_2.imgc
Se inicia el proceso
El archivo D:\Documents\GitHub\PROPCompressorHome\PROPCompressor\EXE\JPEG\JuegosDePruebas\Prueba6\falls_2.imgc se ha descomprimido correctamente!
Ha tardado 8.2865166s y se ha guardado en D:\Documents\GitHub\PROPCompressorHome\PROPCompressor\EXE\JPEG\JuegosDePruebas\Prueba6\falls_2_out.ppm.
El cambio de tamaño pasa de 1169298B a 2332816B con diferencia de 1163518B que resulta en un 199.0% del archivo original.
```

Salida: res Esperado.ppm → Imagen ppm descomprimida.

Veredicto: OK

Trie

Prueba 1

Descripción: Probar el añadido efectivo de caracteres.

Objetivo: Añadir 5 series de secuencias y ver si el índice returned corresponde al del penúltimo byte de la secuencia.

Driver Usado: TrieDriver

Entrada: Introducimos las secuencias: {a, ab, b, abc, c}.

Resultado Esperado: El índice del penúltimo byte de las secuencias debería ser respectivamente {0, 1, 0, 2, 0}

Salida por pantalla:

```
Introduzca uno de los siguientes comandos disponibles:  
1. vaciaArbol  
2. buscaSecuencia  
3. añadeSecuencia  
4. estaLleno?  
5. salir  
3  
Introduzca la secuencia a insertar, toda la secuencia excepto el último byte debe existir previamente en el árbol:  
a  
La secuencia se ha insertado en el árbol. El indice del penúltimo byte es 0  
  
Introduzca uno de los siguientes comandos disponibles:  
1. vaciaArbol  
2. buscaSecuencia  
3. añadeSecuencia  
4. estaLleno?  
5. salir  
3  
Introduzca la secuencia a insertar, toda la secuencia excepto el último byte debe existir previamente en el árbol:  
ab  
La secuencia se ha insertado en el árbol. El indice del penúltimo byte es 1  
  
Introduzca uno de los siguientes comandos disponibles:  
1. vaciaArbol  
2. buscaSecuencia  
3. añadeSecuencia  
4. estaLleno?  
5. salir  
3  
Introduzca la secuencia a insertar, toda la secuencia excepto el último byte debe existir previamente en el árbol:  
b  
La secuencia se ha insertado en el árbol. El indice del penúltimo byte es 0  
  
Introduzca uno de los siguientes comandos disponibles:  
1. vaciaArbol  
2. buscaSecuencia  
3. añadeSecuencia  
4. estaLleno?  
5. salir  
3  
Introduzca la secuencia a insertar, toda la secuencia excepto el último byte debe existir previamente en el árbol:  
abc  
La secuencia se ha insertado en el árbol. El indice del penúltimo byte es 2  
  
Introduzca uno de los siguientes comandos disponibles:  
1. vaciaArbol  
2. buscaSecuencia  
3. añadeSecuencia  
4. estaLleno?  
5. salir  
3  
Introduzca la secuencia a insertar, toda la secuencia excepto el último byte debe existir previamente en el árbol:  
c  
La secuencia se ha insertado en el árbol. El indice del penúltimo byte es 0
```

Salida: Todos los elementos se han podido añadir y el índice del penúltimo byte es correcto.

Veredicto: OK

Prueba 2

Descripción: Probar la correcta búsqueda de caracteres.

Objetivo: Añadir 2 series de secuencias, buscarlas y ver si se encuentran o no en el árbol.

Driver Usado: TrieDriver

Entrada: Introducimos las secuencias: {a, ab}.

Resultado Esperado: Debería respondernos afirmativamente en los dos casos

Salida por pantalla:

```
Introduzca uno de los siguientes comandos disponibles:  
1. vaciaArbol  
2. buscaSecuencia  
3. añadeSecuencia  
4. estaLleno?  
5. salir  
3  
Introduzca la secuencia a insertar, toda la secuencia excepto el último byte debe existir previamente en el árbol:  
a  
La secuencia se ha insertado en el árbol. El índice del penúltimo byte es 0  
  
Introduzca uno de los siguientes comandos disponibles:  
1. vaciaArbol  
2. buscaSecuencia  
3. añadeSecuencia  
4. estaLleno?  
5. salir  
3  
Introduzca la secuencia a insertar, toda la secuencia excepto el último byte debe existir previamente en el árbol:  
ab  
La secuencia se ha insertado en el árbol. El índice del penúltimo byte es 1  
  
Introduzca uno de los siguientes comandos disponibles:  
1. vaciaArbol  
2. buscaSecuencia  
3. añadeSecuencia  
4. estaLleno?  
5. salir  
2  
Introduzca la secuencia a buscar:  
a  
La secuencia se encuentra dentro del árbol.  
  
Introduzca uno de los siguientes comandos disponibles:  
1. vaciaArbol  
2. buscaSecuencia  
3. añadeSecuencia  
4. estaLleno?  
5. salir  
2  
Introduzca la secuencia a buscar:  
ab  
La secuencia se encuentra dentro del árbol.
```

Salida: Todos los elementos se han podido añadir y se han podido encontrar en el árbol.

Veredicto: OK

Prueba 3

Descripción: Probar la correcta búsqueda de caracteres que no existen en el árbol.

Objetivo: Buscar 2 series de secuencias no añadidas al árbol y ver si se encuentran o no en el árbol.

Driver Usado: TrieDriver

Entrada: Buscamos las secuencias: {a, ab}.

Resultado Esperado: Debería respondernos negativamente en los dos casos

Salida por pantalla:

```
Introduzca uno de los siguientes comandos disponibles:
```

1. vaciaArbol
2. buscaSecuencia
3. añadeSecuencia
4. estaLleno?
5. salir

```
2
```

```
Introduzca la secuencia a buscar:
```

```
a
```

```
La secuencia no se encuentra dentro del árbol.
```

```
Introduzca uno de los siguientes comandos disponibles:
```

1. vaciaArbol
2. buscaSecuencia
3. añadeSecuencia
4. estaLleno?
5. salir

```
2
```

```
Introduzca la secuencia a buscar:
```

```
ab
```

```
La secuencia no se encuentra dentro del árbol.
```

Salida: Todos los elementos no se han podido encontrar en el árbol.

Veredicto: OK

Prueba 4

Descripción: Probar correcto vaciado del diccionario.

Objetivo: Ver si después de vaciar el árbol se siguen encontrando los caracteres en el árbol o no.

Driver Usado: TrieDriver

Entrada:

- Introducimos las secuencias: {a, ab}.
- Vaciar árbol.
- Buscar las secuencias: {a, ab}.

Resultado Esperado: Debería respondernos negativamente en los dos casos.

Salida por pantalla:

```

Introduzca uno de los siguientes comandos disponibles:
1. vaciaArbol
2. buscaSecuencia
3. añadeSecuencia
4. estaLleno?
5. salir
3
Introduzca la secuencia a insertar, toda la secuencia excepto el último byte debe existir previamente en el árbol:
a
La secuencia se ha insertado en el árbol. El índice del penúltimo byte es 0

Introduzca uno de los siguientes comandos disponibles:
1. vaciaArbol
2. buscaSecuencia
3. añadeSecuencia
4. estaLleno?
5. salir
3
Introduzca la secuencia a insertar, toda la secuencia excepto el último byte debe existir previamente en el árbol:
ab
La secuencia se ha insertado en el árbol. El índice del penúltimo byte es 1

Introduzca uno de los siguientes comandos disponibles:
1. vaciaArbol
2. buscaSecuencia
3. añadeSecuencia
4. estaLleno?
5. salir
1
El árbol se ha vaciado correctamente.

Introduzca uno de los siguientes comandos disponibles:
1. vaciaArbol
2. buscaSecuencia
3. añadeSecuencia
4. estaLleno?
5. salir
2
Introduzca la secuencia a buscar:
a
La secuencia no se encuentra dentro del árbol.

Introduzca uno de los siguientes comandos disponibles:
1. vaciaArbol
2. buscaSecuencia
3. añadeSecuencia
4. estaLleno?
5. salir
2
Introduzca la secuencia a buscar:
ab
La secuencia no se encuentra dentro del árbol.

```

Salida: Todos los elementos no se han podido encontrar en el árbol después de vaciarlo.

Veredicto: OK

Prueba 5

Descripción: Probar correcto vaciado del diccionario.

Objetivo: Ver si después de vaciar el árbol el contador del índice se ha reiniciado.

Driver Usado: TrieDriver

Entrada:

- Introducimos las secuencias: {a, ab}.
- Vaciamos árbol.
- Introducimos las secuencias: {a, ab}.

Resultado Esperado: Se deberían generar de nuevo las secuencias con exactamente los mismos índices para los penúltimos bytes de las secuencias.

Salida por pantalla:

```

Introduzca uno de los siguientes comandos disponibles:
1. vaciaArbol
2. buscaSecuencia
3. añadeSecuencia
4. estaLleno?
5. salir
3
Introduzca la secuencia a insertar, toda la secuencia excepto el último byte debe existir previamente en el árbol:
a
La secuencia se ha insertado en el árbol. El indice del penúltimo byte es 0

Introduzca uno de los siguientes comandos disponibles:
1. vaciaArbol
2. buscaSecuencia
3. añadeSecuencia
4. estaLleno?
5. salir
3
Introduzca la secuencia a insertar, toda la secuencia excepto el último byte debe existir previamente en el árbol:
ab
La secuencia se ha insertado en el árbol. El indice del penúltimo byte es 1

Introduzca uno de los siguientes comandos disponibles:
1. vaciaArbol
2. buscaSecuencia
3. añadeSecuencia
4. estaLleno?
5. salir
1
El árbol se ha vaciado correctamente.

Introduzca uno de los siguientes comandos disponibles:
1. vaciaArbol
2. buscaSecuencia
3. añadeSecuencia
4. estaLleno?
5. salir
3
Introduzca la secuencia a insertar, toda la secuencia excepto el último byte debe existir previamente en el árbol:
a
La secuencia se ha insertado en el árbol. El indice del penúltimo byte es 0

Introduzca uno de los siguientes comandos disponibles:
1. vaciaArbol
2. buscaSecuencia
3. añadeSecuencia
4. estaLleno?
5. salir
3
Introduzca la secuencia a insertar, toda la secuencia excepto el último byte debe existir previamente en el árbol:
ab
La secuencia se ha insertado en el árbol. El indice del penúltimo byte es 1

```

Salida: Todos los elementos se han podido añadir con el índice correcto.

Veredicto: OK

Prueba 6

Descripción: Probar que el diccionario no está lleno.

Objetivo: Ver que el diccionario no está lleno.

Driver Usado: TrieDriver

Entrada:

- Introducimos las secuencias: {a, ab}.
- Comprobamos si el árbol está lleno.

Resultado Esperado: Debería retornarnos que el árbol no está lleno.

Salida por pantalla:

```
Introduzca uno de los siguientes comandos disponibles:  
1. vaciaArbol  
2. buscaSecuencia  
3. añadeSecuencia  
4. estaLleno?  
5. salir  
3  
Introduzca la secuencia a insertar, toda la secuencia excepto el último byte debe existir previamente en el árbol:  
a  
La secuencia se ha insertado en el árbol. El índice del penúltimo byte es 0  
Introduzca uno de los siguientes comandos disponibles:  
1. vaciaArbol  
2. buscaSecuencia  
3. añadeSecuencia  
4. estaLleno?  
5. salir  
3  
Introduzca la secuencia a insertar, toda la secuencia excepto el último byte debe existir previamente en el árbol:  
ab  
La secuencia se ha insertado en el árbol. El índice del penúltimo byte es 1  
Introduzca uno de los siguientes comandos disponibles:  
1. vaciaArbol  
2. buscaSecuencia  
3. añadeSecuencia  
4. estaLleno?  
5. salir  
4  
El árbol no está lleno, puedes agregar más secuencias de bytes sin problema.
```

Salida: Efectivamente el árbol no está lleno.

Veredicto: OK

ProcesoComprimir

Prueba 1

Descripción: compresión de un archivo de texto vacío con el algoritmo LZSS.

Objetivo: crear un procesoComprimir que comprima un fichero de texto vacío con un cierto algoritmo de compresión de texto.

Driver Usado: ProcesoComprimirDriver

Entrada: vacio.txt → fichero de texto vacío.

Resultado Esperado: creación correcta de un procesoComprimir, compresión correcta del fichero especificado, creación correcta de una instancia de DatosProceso con los datos estadísticos recopilados como resultado de llevar a cabo el proceso de compresión.

Salida por pantalla:

Introduzca uno de los siguientes comandos disponibles:

```
1. comprimir
2. salir
1
Escriba el path relativo del fichero a comprimir (.txt o .ppm!)
EXE/ProcesoComprimir/JuegosDePruebas/Prueba1/vacio.txt
Escriba el algoritmo de compresión que quiera usar, de entre los siguientes:
predeterminado
lzs
lz78
lzw
lzs
Se inicia el proceso
El archivo D:\Documents\GitHub\PROPCompressorHome\PROPCompressor\EXE\ProcesoComprimir\JuegosDePruebas\Prueba1\vacio.txt se ha comprimido correctamente!
Ha tardado 1.077E-4s y se ha guardado en D:\Documents\GitHub\PROPCompressorHome\PROPCompressor\EXE\ProcesoComprimir\JuegosDePruebas\Prueba1\vacio.lzs.
El cambio de tamaño pasa de 0B a 4B con diferencia de -4B que resulta en un Infinity% del archivo original.
```

Salida: res Esperado.lzs → fichero de texto que contiene los datos de vacio.txt en un formato comprimido sin pérdidas.

Veredicto: OK

Prueba 2

Descripción: compresión de un archivo de texto vacío con el algoritmo LZW.

Objetivo: crear un procesoComprimir que comprima un fichero de texto vacío con un cierto algoritmo de compresión de texto.

Driver Usado: ProcesoComprimirDriver

Entrada: vacio.txt → fichero de texto vacío.

Resultado Esperado: creación correcta de un procesoComprimir, compresión correcta del fichero especificado, creación correcta de una instancia de DatosProceso con los datos estadísticos recopilados como resultado de llevar a cabo el proceso de compresión.

Salida por pantalla:

Introduzca uno de los siguientes comandos disponibles:

```
1. comprimir
2. salir
1
Escriba el path relativo del fichero a comprimir (.txt o .ppm!)
EXE/ProcesoComprimir/JuegosDePruebas/Prueba2/vacio.txt
Escriba el algoritmo de compresión que quiera usar, de entre los siguientes:
predeterminado
lzss
lz78
lzw
lzw
Se inicia el proceso
El archivo D:\Documents\GitHub\PROPCompressorHome\PROPCompressor\EXE/ProcesoComprimir/JuegosDePruebas/Prueba2/vacio.txt se ha comprimido correctamente!
Ha tardado 5.823E-4s y se ha guardado en D:\Documents\GitHub\PROPCompressorHome\PROPCompressor\EXE/ProcesoComprimir/JuegosDePruebas/Prueba2/vacio.lzw.
El cambio de tamaño pasa de 0B a 0B con diferencia de 0B que resulta en un 100.0% del archivo original.
```

Salida: res Esperado.lzw → fichero de texto que contiene los datos de vacio.txt en un formato comprimido sin pérdidas.

Veredicto: OK

Prueba 3

Descripción: compresión de un archivo de texto vacío con el algoritmo LZ78.

Objetivo: crear un procesoComprimir que comprima un fichero de texto vacío con un cierto algoritmo de compresión de texto.

Driver Usado: ProcesoComprimirDriver

Entrada: vacio.txt → fichero de texto vacío.

Resultado Esperado: creación correcta de un procesoComprimir, compresión correcta del fichero especificado, creación correcta de una instancia de DatosProceso con los datos estadísticos recopilados como resultado de llevar a cabo el proceso de compresión.

Salida por pantalla:

Introduzca uno de los siguientes comandos disponibles:

```
1. comprimir
2. salir
1
Escriba el path relativo del fichero a comprimir (.txt o .ppm!)
EXE/ProcesoComprimir/JuegosDePruebas/Prueba3/vacio.txt
Escriba el algoritmo de compresión que quiera usar, de entre los siguientes:
predeterminado
lzss
lz78
lzw
Lz78
Se inicia el proceso
El archivo D:\Documents\GitHub\PROPCompressorHome\PROPCompressor\EXE/ProcesoComprimir/JuegosDePruebas/Prueba3/vacio.txt se ha comprimido correctamente!
Ha tardado 0.0034837s y se ha guardado en D:\Documents\GitHub\PROPCompressorHome\PROPCompressor\EXE/ProcesoComprimir/JuegosDePruebas/Prueba3/vacio.lz78.
El cambio de tamaño pasa de 0B a 0B con diferencia de 0B que resulta en un 100.0% del archivo original.
```

Salida: res Esperado.lz78 → fichero de texto que contiene los datos de vacio.txt en un formato comprimido sin pérdidas.

Veredicto: OK

Prueba 4

Descripción: compresión de un archivo de imagen vacío con el algoritmo JPEG.

Objetivo: crear un procesoComprimir que comprima un fichero de imagen vacío con el algoritmo JPEG.

Driver Usado: ProcesoComprimirDriver

Entrada: void.ppm → fichero de imagen vacío.

Resultado Esperado: creación correcta de un procesoComprimir, activación y tratado de excepción de formato erróneo.

Salida por pantalla:

```
Introduzca uno de los siguientes comandos disponibles:
```

```
1. comprimir
2. salir
1
Escriba el path relativo del fichero a comprimir (.txt o .ppm!)
EXE/ProcesoComprimir/JuegosDePruebas/Prueba4/void.ppm
Indique la calidad de compresión a usar (del 1 al 7)
4
Se inicia el proceso
El formato de .ppm no es correcto!
```

Salida:

Veredicto: OK

Prueba 5

Descripción: compresión de un archivo de imagen con errores con el algoritmo JPEG.

Objetivo: crear un procesoComprimir que comprima un fichero de imagen con errores de formato con el algoritmo JPEG.

Driver Usado: ProcesoComprimirDriver

Entrada: wrong.ppm → fichero de imagen con errores en la cabecera e inconsistencias de valores en el pixel map respecto a las dimensiones.

Resultado Esperado: creación correcta de un procesoComprimir, activación y tratado de excepción de formato erróneo.

Salida por pantalla:

```
Introduzca uno de los siguientes comandos disponibles:
```

```
1. comprimir
2. salir
1
Escriba el path relativo del fichero a comprimir (.txt o .ppm!)
EXE/ProcesoComprimir/JuegosDePruebas/Prueba5/wrong.ppm
Indique la calidad de compresión a usar (del 1 al 7)
4
Se inicia el proceso
El formato de .ppm no es correcto!
```

Salida:

Veredicto: OK

Prueba 6

Descripción: compresión de un archivo de texto con el algoritmo LZSS.

Objetivo: crear un procesoComprimir que comprima un cierto fichero de texto con un cierto algoritmo de compresión de texto.

Driver Usado: ProcesoComprimirDriver

Entrada: el_quijote.txt → fichero de texto que contiene una secuencia extensa y compleja de caracteres.

Resultado Esperado: creación correcta de un procesoComprimir, compresión correcta del fichero especificado, creación correcta de una instancia de DatosProceso con los datos estadísticos recopilados como resultado de llevar a cabo el proceso de compresión.

Salida por pantalla:

Introduzca uno de los siguientes comandos disponibles:

```
1. comprimir
2. salir
1
Escriba el path relativo del fichero a comprimir (.txt o .ppm!)
EXE/ProcesoComprimir/JuegosDePruebas/Prueba6/el_quijote.txt
Escriba el algoritmo de compresión que quiera usar, de entre los siguientes:
predeterminado
lzss
lz78
lzw
Lzss
Se inicia el proceso
El archivo D:\Documents\GitHub\PROPCompressorHome\PROPCompressor\EXE/ProcesoComprimir/JuegosDePruebas/Prueba6/el_quijote.txt se ha comprimido correctamente!
Ha tardado 4.7630566s y se ha guardado en D:\Documents\GitHub\PROPCompressorHome\PROPCompressor\EXE/ProcesoComprimir/JuegosDePruebas/Prueba6/el_quijote.lzss.
El cambio de tamaño pasa de 10624458 a 5118058 con diferencia de 550640B que resulta en un 48.0% del archivo original.
```

Salida: res Esperado.lzss → fichero de texto que contiene los datos de el_quijote.txt en un formato comprimido sin pérdidas.

Veredicto: OK

Prueba 7

Descripción: compresión de un archivo de texto con el algoritmo LZW.

Objetivo: crear un procesoComprimir que comprima un cierto fichero de texto con un cierto algoritmo de compresión de texto.

Driver Usado: ProcesoComprimirDriver

Entrada: el_quijote.txt → fichero de texto que contiene una secuencia extensa y compleja de caracteres.

Resultado Esperado: creación correcta de un procesoComprimir, compresión correcta del fichero especificado, creación correcta de una instancia de DatosProceso con los datos estadísticos recopilados como resultado de llevar a cabo el proceso de compresión.

Salida por pantalla:

Introduzca uno de los siguientes comandos disponibles:

```
1. comprimir
2. salir
1
Escriba el path relativo del fichero a comprimir (.txt o .ppm!)
EXE/ProcesoComprimir/JuegosDePruebas/Prueba7/el_quijote.txt
Escriba el algoritmo de compresión que quiera usar, de entre los siguientes:
predeterminado
lzss
lz78
lzw
Lzw
Se inicia el proceso
El archivo D:\Documents\GitHub\PROPCompressorHome\PROPCompressor\EXE/ProcesoComprimir/JuegosDePruebas/Prueba7/el_quijote.txt se ha comprimido correctamente!
Ha tardado 0.5129721s y se ha guardado en D:\Documents\GitHub\PROPCompressorHome\PROPCompressor\EXE/ProcesoComprimir/JuegosDePruebas/Prueba7/el_quijote.lzw.
El cambio de tamaño pasa de 10624458 a 7305768 con diferencia de 331869B que resulta en un 68.0% del archivo original.
```

Salida: res Esperado.lzw → fichero de texto que contiene los datos de el_quijote.txt en un formato comprimido sin pérdidas.

Veredicto: OK

Prueba 8

Descripción: compresión de un archivo de texto con el algoritmo LZ78.

Objetivo: crear un procesoComprimir que comprima un cierto fichero de texto con un cierto algoritmo de compresión de texto.

Driver Usado: ProcesoComprimirDriver

Entrada: el_quijote.txt → fichero de texto que contiene una secuencia extensa y compleja de caracteres.

Resultado Esperado: creación correcta de un procesoComprimir, compresión correcta del fichero especificado, creación correcta de una instancia de DatosProceso con los datos estadísticos recopilados como resultado de llevar a cabo el proceso de compresión.

Salida por pantalla:

Introduzca uno de los siguientes comandos disponibles:

```
1. comprimir
2. salir
1
Escriba el path relativo del fichero a comprimir (.txt o .ppm)
EXE/ProcesoComprimir/JuegosDePruebas/Prueba8/el_quijote.txt
Escriba el algoritmo de compresión que quiera usar, de entre los siguientes:
predeterminado
lzs
lz78
lzw
lz78
Se inicia el proceso
El archivo D:\Documents\GitHub\PROPCompressorHome\PROPCompressor\EXE\ProcesoComprimir\JuegosDePruebas\Prueba8\el_quijote.txt se ha comprimido correctamente!
Ha tardado 0.2593982s y se ha guardado en D:\Documents\GitHub\PROPCompressorHome\PROPCompressor\EXE\ProcesoComprimir\JuegosDePruebas\Prueba8\el_quijote.lz78.
El cambio de tamaño pasa de 10624458 a 5729758 con diferencia de 4894708 que resulta en un 53.0% del archivo original.
```

Salida: res Esperado.lz78 → fichero de texto que contiene los datos de el_quijote.txt en un formato comprimido sin pérdidas.

Veredicto: OK

Prueba 9

Descripción: compresión de un archivo de imagen con el algoritmo JPEG.

Objetivo: crear un procesoComprimir que comprima un cierto fichero de imagen con el algoritmo JPEG.

Driver Usado: ProcesoComprimirDriver

Entrada: falls_2.ppm → un fichero de imagen ppm con magicNumber P6, dimensiones positivas y rgbMaxVal con valor 255. También se espera una calidad de compresión, del 1 al 7, siendo 1 la más baja posible y 7 la más alta (se usa 4, pero se podría usar cualquier otro valor dentro del rango a voluntad).

Resultado Esperado: creación correcta de un procesoComprimir, compresión correcta del fichero especificado, creación correcta de una instancia de DatosProceso con los datos estadísticos recopilados como resultado de llevar a cabo el proceso de compresión.

Salida por pantalla:

Introduzca uno de los siguientes comandos disponibles:

```
1. comprimir
2. salir
1
Escriba el path relativo del fichero a comprimir (.txt o .ppm!)
EXE/ProcesoComprimir/JuegosDePruebas/Prueba9/falls_2.ppm
Indique la calidad de compresión a usar (del 1 al 7)
5
Se inicia el proceso
El archivo D:\Documents\GitHub\PROPCompressorHome\PROPCompressor\EXE/ProcesoComprimir/JuegosDePruebas/Prueba9/falls_2.ppm se ha comprimido correctamente!
Ha tardado 7.7035085s y se ha guardado en D:\Documents\GitHub\PROPCompressorHome\PROPCompressor\EXE/ProcesoComprimir/JuegosDePruebas/Prueba9/falls_2.imgc.
El cambio de tamaño pasa de 2332861B a 1169298B con diferencia de 1163563B que resulta en un 50.0% del archivo original.
```

Salida: res Esperado.imgc → fichero de imagen que contiene los datos de falls_2.ppm en un formato comprimido con pérdidas. Esto es, una aproximación de la imagen original.

Veredicto: OK

ProcesoDescomprimir

Prueba 1

Descripción: descompresión de un archivo de texto vacío comprimido con el algoritmo LZSS.

Objetivo: crear un procesoDescomprimir que descomprima un fichero de texto vacío comprimido con extensión .lzss.

Driver Usado: ProcesoDescomprimirDriver

Entrada: vacio.lzss → fichero de texto vacío comprimido con el algoritmo LZSS.

Resultado Esperado: creación correcta de un procesoDescomprimir, activación y tratado de excepción de formato erróneo.

Salida por pantalla:

Introduzca uno de los siguientes comandos disponibles:

```
1. descomprimir
2. salir
1
Escriba el path relativo del fichero a descomprimir (.lzss, .lz78, .lzw o .imgc!)
EXE/ProcesoDescomprimir/JuegosDePruebas/Prueba1/vacio.lzss
Se inicia el proceso
El archivo a descomprimir está corrupto o no se ha generado adecuadamente.
```

Salida:

Veredicto: OK

Prueba 2

Descripción: descompresión de un archivo de texto vacío comprimido con el algoritmo LZW.

Objetivo: crear un procesoDescomprimir que descomprima un fichero de texto vacío comprimido con extensión .lzw.

Driver Usado: ProcesoDescomprimirDriver

Entrada: vacio.lzw → fichero de texto vacío comprimido con el algoritmo LZW.

Resultado Esperado: creación correcta de un procesoDescomprimir, descompresión correcta del fichero especificado, creación correcta de una instancia de DatosProceso con los datos estadísticos recopilados como resultado de llevar a cabo el proceso de descompresión.

Salida por pantalla:

Introduzca uno de los siguientes comandos disponibles:

```
1. descomprimir
2. salir
1
Escriba el path relativo del fichero a descomprimir (.lzss, .lz78, .lzw o .imgc!)
EXE/ProcesoDescomprimir/JuegosDePruebas/Prueba2/vacio.lzw
Se inicia el proceso
El archivo D:\Documents\GitHub\PROPCompressorHome\PROPCompressor\EXE/ProcesoDescomprimir/JuegosDePruebas/Prueba2/vacio.lzw se ha descomprimido correctamente!
Ha tardado 6.069E-4s y se ha guardado en D:\Documents\GitHub\PROPCompressorHome\PROPCompressor\EXE/ProcesoDescomprimir/JuegosDePruebas/Prueba2/vacio_out.txt.
El cambio de tamaño pasa de 0B a 0B con diferencia de 0B que resulta en un 100.0% del archivo original.
```

Salida: res Esperado.txt → fichero de texto que contiene los datos de vacio.lzw en un formato descomprimido sin pérdidas.

Veredicto: OK

Prueba 3

Descripción: descompresión de un archivo de texto vacío comprimido con el algoritmo LZ78.

Objetivo: crear un procesoDescomprimir que descomprima un fichero de texto vacío comprimido con extensión .lz78.

Driver Usado: ProcesoDescomprimirDriver

Entrada: vacio.lz78 → fichero de texto vacío comprimido con el algoritmo LZ78.

Resultado Esperado: creación correcta de un procesoDescomprimir, descompresión correcta del fichero especificado, creación correcta de una instancia de DatosProceso con los datos estadísticos recopilados como resultado de llevar a cabo el proceso de descompresión.

Salida por pantalla:

Introduzca uno de los siguientes comandos disponibles:

```
1. descomprimir
2. salir
1
Escriba el path relativo del fichero a descomprimir (.lzs, .lz78, .lzw o .imgc!)
EXE/ProcesoDescomprimir/JuegosDePruebas/Prueba3/vacio.lz78
Se inició el proceso
El archivo D:\Documents\GitHub\PROPCompressorHome\PROPCompressor\EXE\ProcesoDescomprimir\JuegosDePruebas\Prueba3\vacio.lz78 se ha descomprimido correctamente!
Ha tardado 1.92E-5s y se ha guardado en D:\Documents\GitHub\PROPCompressorHome\PROPCompressor\EXE\ProcesoDescomprimir\JuegosDePruebas\Prueba3\vacio_out.txt.
El cambio de tamaño pasa de 0B a 0B con diferencia de 0B que resulta en un 100.0% del archivo original.
```

Salida: res Esperado.txt → fichero de texto que contiene los datos de vacio.lz78 en un formato descomprimido sin pérdidas.

Veredicto: OK

Prueba 4

Descripción: descompresión de un archivo de imagen vacío comprimido con el algoritmo JPEG.

Objetivo: crear un procesoDescomprimir que descomprima un fichero de imagen vacío con el algoritmo JPEG.

Driver Usado: ProcesoDescomprimirDriver

Entrada: void.imgc → fichero de imagen comprimida vacío.

Resultado Esperado: creación correcta de un procesoDescomprimir, activación y tratado de excepción de formato erróneo.

Salida por pantalla:

Introduzca uno de los siguientes comandos disponibles:

```
1. descomprimir
2. salir
1
Escriba el path relativo del fichero a descomprimir (.lzss, .lz78, .lzw o .imgc!)
EXE/ProcesoDescomprimir/JuegosDePruebas/Prueba4/void.imgc
Se inicia el proceso
El formato de .imgc no es correcto!
```

Salida:

Veredicto: OK

Prueba 5

Descripción: descompresión de un archivo de texto con errores con el algoritmo LZSS.

Objetivo: crear un procesoDescomprimir que descomprima un fichero de texto con errores comprimido con extensión .lzss.

Driver Usado: ProcesoDescomprimirDriver

Entrada: Garbage.lzss → archivo generado a mano.

Resultado Esperado: creación correcta de un procesoDescomprimir, activación y tratado de excepción de formato erróneo.

Salida por pantalla:

Introduzca uno de los siguientes comandos disponibles:

```
1. descomprimir
2. salir
1
Escriba el path relativo del fichero a descomprimir (.lzss, .lz78, .lzw o .imgc!)
EXE/ProcesoDescomprimir/JuegosDePruebas/Prueba5/Garbage.lzss
Se inicia el proceso
El archivo a descomprimir está corrupto o no se ha generado adecuadamente.
```

Salida:

Veredicto: OK

Prueba 6

Descripción: descompresión de un archivo de texto con errores con el algoritmo LZW.

Objetivo: crear un procesoDescomprimir que descomprima un fichero de texto con errores comprimido con extensión .lzw.

Driver Usado: ProcesoDescomprimirDriver

Entrada: Garbage.lzw → archivo generado a mano.

Resultado Esperado: creación correcta de un procesoDescomprimir, activación y tratado de excepción de formato erróneo.

Salida por pantalla:

Introduzca uno de los siguientes comandos disponibles:

```
1. descomprimir
2. salir
1
Escriba el path relativo del fichero a descomprimir (.lzss, .lz78, .lzw o .imgc!)
EXE/ProcesoDescomprimir/JuegosDePruebas/Prueba6/Garbage.lzw
Se inicia el proceso
El archivo a descomprimir está corrupto o no se ha generado adecuadamente.
```

Salida:

Veredicto: OK

Prueba 7

Descripción: descompresión de un archivo de texto con errores con el algoritmo LZ78.

Objetivo: crear un procesoDescomprimir que descomprima un fichero de texto con errores comprimido con extensión .lz78.

Driver Usado: ProcesoDescomprimirDriver

Entrada: Garbage.lz78 → archivo generado a mano.

Resultado Esperado: creación correcta de un procesoDescomprimir, activación y tratado de excepción de formato erróneo.

Salida por pantalla:

Introduzca uno de los siguientes comandos disponibles:

```
1. descomprimir
2. salir
1
Escriba el path relativo del fichero a descomprimir (.lzss, .lz78, .lzw o .imgc!)
EXE/ProcesoDescomprimir/JuegosDePruebas/Prueba7/Garbage.lz78
Se inicia el proceso
El archivo a descomprimir está corrupto o no se ha generado adecuadamente.
```

Salida:

Veredicto: OK

Prueba 8

Descripción: descompresión de un archivo de imagen con errores en el tanto en el header como en el pixel map comprimido con el algoritmo JPEG.

Objetivo: crear un procesoDescomprimir que descomprima un fichero de imagen con errores de formato tanto en el header como en el pixel map comprimido con el algoritmo JPEG.

Driver Usado: ProcesoDescomprimirDriver

Entrada: wrong.ppm → fichero de imagen con errores en la cabecera e inconsistencias de valores en el pixel map respecto a las dimensiones.

Resultado Esperado: creación correcta de un procesoDescomprimir, activación y tratado de excepción de formato erróneo.

Salida por pantalla:

Introduzca uno de los siguientes comandos disponibles:

```
1. descomprimir
2. salir
1
Escriba el path relativo del fichero a descomprimir (.lzss, .lz78, .lzw o .imgc!)
EXE/ProcesoDescomprimir/JuegosDePruebas/Prueba8/wrong.imgc
Se inicia el proceso
El formato de .imgc no es correcto!
```

Salida:

Veredicto: OK

Prueba 9

Descripción: descompresión de un archivo de texto comprimido con el algoritmo LZSS.

Objetivo: crear un procesoDescomprimir que descomprima un fichero de texto comprimido con extensión .lzss.

Driver Usado: ProcesoDescomprimirDriver

Entrada: el_quijote.lzss → fichero de texto que contiene una secuencia extensa y compleja de caracteres comprimidos.

Resultado Esperado: creación correcta de un procesoDescomprimir, descompresión correcta del fichero especificado, creación correcta de una instancia de DatosProceso con los datos estadísticos recopilados como resultado de llevar a cabo el proceso de descompresión.

Salida por pantalla:

Introduzca uno de los siguientes comandos disponibles:

```
1. descomprimir
2. salir
1
Escriba el path relativo del fichero a descomprimir (.lzss, .lz78, .lzw o .imgc!)
EXE/ProcesoDescomprimir/JuegosDePruebas/Prueba9/el_quijote.lzss
Se inicia el proceso
El archivo D:\Documents\GitHub\PROPCompressorHome\PROPCompressor\EXE/ProcesoDescomprimir/JuegosDePruebas/Prueba9/el_quijote.lzss se ha descomprimido correctamente!
Ha tardado 0.0520062s y se ha guardado en D:\Documents\GitHub\PROPCompressorHome\PROPCompressor\EXE/ProcesoDescomprimir/JuegosDePruebas/Prueba9/el_quijote_out.txt.
El cambio de tamaño pasa de 511805B a 1062445B con diferencia de 550640B que resulta en un 207.0% del archivo original.
```

Salida: res Esperado.txt → fichero de texto que contiene los datos de el_quijote.lzss en un formato comprimido sin pérdidas.

Veredicto: OK

Prueba 10

Descripción: descompresión de un archivo de texto comprimido con el algoritmo LZW.

Objetivo: crear un procesoDescomprimir que descomprima un fichero de texto comprimido con extensión .lzw.

Driver Usado: ProcesoDescomprimirDriver

Entrada: el_quijote.lzw → fichero de texto que contiene una secuencia extensa y compleja de caracteres comprimidos.

Resultado Esperado: creación correcta de un procesoDescomprimir, descompresión correcta del fichero especificado, creación correcta de una instancia de DatosProceso con los datos estadísticos recopilados como resultado de llevar a cabo el proceso de descompresión.

Salida por pantalla:

Introduzca uno de los siguientes comandos disponibles:

```
1. descomprimir
2. salir
1
Escriba el path relativo del fichero a descomprimir (.lzss, .lz78, .lzw o .imgc!)
EXE/ProcesoDescomprimir/JuegosDePruebas/Prueba10/el_quijote.lzw
Se inicia el proceso
El archivo D:\Documents\GitHub\PROPCompressorHome\PROPCompressor\EXE\ProcesoDescomprimir\JuegosDePruebas\Prueba10\el_quijote.lzw se ha descomprimido correctamente!
Ha tardado 0.3583657s y se ha guardado en D:\Documents\GitHub\PROPCompressorHome\PROPCompressor\EXE\ProcesoDescomprimir\JuegosDePruebas\Prueba10\el_quijote_out.txt.
El cambio de tamaño pasa de 730576B a 10624458 con diferencia de 331869B que resulta en un 145.0% del archivo original.
```

Salida: res Esperado.txt → fichero de texto que contiene los datos de el_quijote.lzw en un formato comprimido sin pérdidas.

Veredicto: OK

Prueba 11

Descripción: descompresión de un archivo de texto comprimido con el algoritmo LZ78.

Objetivo: crear un procesoDescomprimir que descomprima un cierto fichero de texto con un cierto algoritmo de compresión de texto.

Driver Usado: ProcesoDescomprimirDriver

Entrada: el_quijote.lz78 → fichero de texto que contiene una secuencia extensa y compleja de caracteres.

Resultado Esperado: creación correcta de un procesoDescomprimir, descompresión correcta del fichero especificado, creación correcta de una instancia de DatosProceso con los datos estadísticos recopilados como resultado de llevar a cabo el proceso de descompresión.

Salida por pantalla:

Introduzca uno de los siguientes comandos disponibles:

```
1. descomprimir
2. salir
1
Escriba el path relativo del fichero a descomprimir (.lzss, .lz78, .lzw o .imgc!)
EXE/ProcesoDescomprimir/JuegosDePruebas/Prueba11/el_quijote.lz78
Se inicia el proceso
El archivo D:\Documents\GitHub\PROPCompressorHome\PROPCompressor\EXE\ProcesoDescomprimir\JuegosDePruebas\Prueba11\el_quijote.lz78 se ha descomprimido correctamente!
Ha tardado 0.3780164s y se ha guardado en D:\Documents\GitHub\PROPCompressorHome\PROPCompressor\EXE\ProcesoDescomprimir\JuegosDePruebas\Prueba11\el_quijote_out.txt.
El cambio de tamaño pasa de 572975B a 10624458 con diferencia de 489470B que resulta en un 185.0% del archivo original.
```

Salida: res Esperado.txt → fichero de texto que contiene los datos de el_quijote.lz78 en un formato comprimido sin pérdidas.

Veredicto: OK

Prueba 12

Descripción: descompresión de un archivo de imagen comprimido con el algoritmo JPEG.

Objetivo: crear un procesoDescomprimir que descomprima un cierto fichero de imagen comprimido con el algoritmo JPEG.

Driver Usado: ProcesoDescomprimirDriver

Entrada: falls_2.imgc → un fichero de imagen ppm con magicNumber P6, dimensiones positivas, rgbMaxVal con valor 255 y calidad con valor del 1 al 7.

Resultado Esperado: creación correcta de un procesoDescomprimir, descompresión correcta del fichero especificado, creación correcta de una instancia de DatosProceso con los datos estadísticos recopilados como resultado de llevar a cabo el proceso de descompresión.

Salida por pantalla:

Introduzca uno de los siguientes comandos disponibles:

```
1. descomprimir
2. salir
1
Escriba el path relativo del fichero a descomprimir (.lzss, .lz78, .lzw o .imgc!)
EXE/ProcesoDescomprimir/JuegosDePruebas/Prueba12/falls_2.imgc
Se inicia el proceso
El archivo D:\Documents\GitHub\PROPCompressorHome\PROPCompressor\EXE\ProcesoDescomprimir\JuegosDePruebas\Prueba12/falls_2.imgc se ha descomprimido correctamente!
Ha tardado 7.8730166s y se ha guardado en D:\Documents\GitHub\PROPCompressorHome\PROPCompressor\EXE\ProcesoDescomprimir\JuegosDePruebas\Prueba12/falls_2_out.ppm.
El cambio de tamaño pasa de 11692988 a 2332816B con diferencia de 1163518B / 199.0%
```

Salida: res Esperado.ppm → fichero de imagen que contiene los datos de falls_2.imgc en un formato comprimido con pérdidas. Esto es, una aproximación de la imagen original.

Veredicto: OK

CtrlProcesos

Prueba 1

Descripción: Probar el método de compresión de CtrlProcesos con un archivo de texto .txt usando el algoritmo predeterminado sin haberlo modificado.

Objetivo: Comprobar que se realiza la compresión de un archivo .txt usando el algoritmo predeterminado sin haberlo modificado.

Driver Usado: CtrlProcesosDriver

Entrada: cansExtended.txt → fichero que contiene 138 bytes que son caracteres ASCII y ASCIIextended.

Resultado Esperado: res Esperado.lzss → El resultado esperado de la compresión es un archivo .lzss que contiene el resultado de comprimir cansExtended.txt ya que LZSS es el algoritmo por defecto inicial.

Salida por pantalla:

```
Introduzca uno de los siguientes comandos disponibles:
```

```
1. comprimir
2. descomprimir
3. comprimirYdescomprimir
4. cambiarAlgoritmoTextoPredeterminado
5. salir
1
Escriba el path relativo del fichero a comprimir (.txt o .ppm!)
JuegosDePruebas/Pruebal/cansExtended.txt
Escriba el algoritmo de compresión que quiera usar, de entre los siguientes:
predeterminado
lzss
lz78
lzw
predeterminado
Se inicia el proceso
El proceso ha tardado 0.003273626s. El cambio de tamaño pasa de 138B a 70B con diferencia de 68B que resulta en un 50.0% del archivo original.
El archivo /home/jan/Documents/PROP/PROPCompressor/EXE/CtrlProcesos/JuegosDePruebas/Pruebal/cansExtended.txt se ha comprimido correctamente!
```

Salida: cansExtended.lzss → tiene el mismo contenido que el resultado esperado

Veredicto: OK

Prueba 2

Descripción: Probar el método de compresión de CtrlProceso con un archivo .ppm usando el algoritmo JPEG, que es el único y por defecto.

Objetivo: Comprobar que se realiza la compresión de un archivo .ppm usando el algoritmo predeterminado sin haberlo modificado que será .JPEG.

Driver Usado: CtrlProcesosDriver

Entrada: cayuga_1.ppm → imagen en ppm.

Resultado Esperado: res Esperado.imgc → archivo de imagen comprimida.

Salida por pantalla:

```
Introduzca uno de los siguientes comandos disponibles:
```

```
1. comprimir
2. descomprimir
3. comprimirYdescomprimir
4. cambiarAlgoritmoTextoPredeterminado
5. salir
1
Escriba el path relativo del fichero a comprimir (.txt o .ppm!)
JuegosDePruebas/Prueba2/cayuga_1.ppm
Indique la calidad de compresión a usar (del 1 al 7)
4
Se inicia el proceso
El proceso ha tardado 1.777790637s. El cambio de tamaño pasa de 1440060B a 723217B con diferencia de 716843B que resulta en un 50.0% del archivo original.
El archivo /home/jan/Documents/PROP/PROPCompressor/EXE/CtrlProcesos/JuegosDePruebas/Prueba2/cayuga_1.ppm se ha comprimido correctamente!
```

Salida: cayuga_1.imgc → imagen comprimida, con resultado no visible hasta la descompresión.

Veredicto: OK

Prueba 3

Descripción: Probar el método de comprimir de CtrlProceso con un archivo de texto .txt usando un algoritmo de compresión de texto y no el predeterminado.

Objetivo: Comprobar que se realiza la compresión de un archivo .txt usando el algoritmo .lz78, es el mismo caso para lzw.

Driver Usado: CtrlProcesosDriver

Entrada: el_quijote.txt → fichero que contiene 1.060.259 bytes bytes que son caracteres ASCII y ASCIIextended.

Resultado Esperado: res Esperado.lz78 → El resultado esperado de la compresión es un archivo .lz78 que contiene el resultado de comprimir el_quijote.txt ya que LZ78 es el algoritmo por defecto inicial.

Salida por pantalla:

Introduzca uno de los siguientes comandos disponibles:

```
1. comprimir
2. descomprimir
3. comprimirYdescomprimir
4. cambiarAlgoritmoTextoPredeterminado
5. salir
1
Escriba el path relativo del fichero a comprimir (.txt o .ppm!)
JuegosDePruebas/Prueba3/el_quijote.txt
Escriba el algoritmo de compresión que quiera usar, de entre los siguientes:
predeterminado
lzss
lzw
lz78
Se inicia el proceso
El proceso ha tardado 0.458379969s. El cambio de tamaño pasa de 1060259B a 571982B con diferencia de 488277B que resulta en un 53.0% del archivo original.
El archivo /home/jan/Documents/PROP/PROPCompressor/EXE/CtrlProcesos/JuegosDePruebas/Prueba3/el_quijote.txt se ha comprimido correctamente!
```

Salida: el_quijote.lz78 → tiene el mismo contenido que el resultado esperado

Veredicto: OK

Prueba 4

Descripción: Probar el método de descompresión de CtrlProcesos con un archivo de texto comprimido en lzss.

Objetivo: Comprobar que se realiza la descompresión de un archivo .lzss usando el algoritmo correspondiente que detecta el al descomprimir, la detección del algoritmo se prueba para este caso pero es la misma para los otros algoritmos de texto.

Driver Usado: CtrlProcesosDriver

Entrada: cansExtended.lzss → fichero resultante de una compresión con lzss.

Resultado Esperado: res Esperado.txt → resultado de descomprimir con el descompresor de LZSS el archivo inicial.

Salida por pantalla:

Introduzca uno de los siguientes comandos disponibles:

```
1. comprimir
2. descomprimir
3. comprimirYdescomprimir
4. cambiarAlgoritmoTextoPredeterminado
5. salir
2
Escriba el path relativo del fichero a descomprimir (.lzss, .lz78, lzw o .imgc!)
JuegosDePruebas/Prueba4/cansExtended.lzss
Se inicia el proceso
El proceso ha tardado 3.05985E-4s. El cambio de tamaño pasa de 70B a 138B con diferencia de 68B que resulta en un 197.0% del archivo original.
El archivo se ha descomprimido correctamente!
```

Salida: cansExtended_out.txt → tiene el mismo contenido que el resultado esperado.

Veredicto: OK

Prueba 5

Descripción: Probar el método de descompresión de CtrlProceso con un archivo .imgc usando el algoritmo JPEG, que es el único y por defecto.

Objetivo: Comprobar que se realiza la descompresión de un archivo .imgc usando el algoritmo predeterminado sin haberlo modificado que será .JPEG.

Driver Usado: CtrlProcesosDriver

Entrada: cayuga_1.imgc → imagen en imgc.

Resultado Esperado: cayuga_1_out.ppm → archivo de imagen descomprimida.

Salida por pantalla:

Introduzca uno de los siguientes comandos disponibles:

```
1. comprimir
2. descomprimir
3. comprimirydescomprimir
4. cambiarAlgoritmoTextoPredeterminado
5. salir
2
Escriba el path relativo del fichero a descomprimir (.lzss, .lz78, lzw o .imgc!)
JuegosDePruebas/Prueba5/cayuga_1.imgc
Se inicia el proceso
El proceso ha tardado 1.943460413s. El cambio de tamaño pasa de 723217B a 1440015B con diferencia de 716798B que resulta en un 199.0% del archivo original.
El archivo se ha descomprimido correctamente!
```

Salida: cayuga_1.imgc → imagen comprimida, con resultado no visible hasta la descompresión.

Veredicto: OK

Prueba 6

Descripción: Probar el método de comprimir y descomprimir de CtrlProceso con un archivo de texto .txt usando un algoritmo de compresión de texto.

Objetivo: Comprobar que se realiza la compresión de un archivo .txt usando el algoritmo .lz78, es el mismo caso para lzss y lzw.

Driver Usado: CtrlProcesosDriver

Entrada: el_quijote.txt → fichero que contiene 1.060.259 bytes que son caracteres ASCII y ASCIilexended.

Resultado Esperado: res Esperado.txt → resultado de descomprimir el fichero comprimido que inicialmente era .txt.

Salida por pantalla:

Introduzca uno de los siguientes comandos disponibles:

```
1. comprimir
2. descomprimir
3. comprimirydescomprimir
4. cambiarAlgoritmoTextoPredeterminado
5. salir
3
Escriba el path relativo del fichero a comprimir (.txt o .ppm!)
JuegosDePruebas/Prueba6/el_quijote.txt
Escriba el algoritmo de compresión que quiera usar, de entre los siguientes:
predeterminado
lzss
lz78
lzw
lz78
Se inicia el proceso
El proceso ha tardado 0.273780096s. El cambio de tamaño pasa de 1060259B a 571982B con diferencia de 488277B que resulta en un 53.0% del archivo original.
El proceso ha tardado 0.155843829s. El cambio de tamaño pasa de 571982B a 1060259B con diferencia de 488277B que resulta en un 185.0% del archivo original.
El archivo /home/jan/Documents/PROP/PROPCompressor/EXE/CtrlProcesos/JuegosDePruebas/Prueba6/el_quijote.txt se ha comprimido y descomprimido correctamente!
```

Salida: el_quijote_out.txt → tiene el mismo contenido que el resultado esperado

Veredicto: OK

Prueba 7

Descripción: Probar el método de compresión y descompresión de CtrlProceso con un archivo .ppm usando el algoritmo JPEG, que es el único y por defecto.

Objetivo: Comprobar que se realiza la compresión y posterior descompresión de un archivo .ppm usando el algoritmo JPEG.

Driver Usado: CtrlProcesosDriver

Entrada: cayuga_1.ppm → imagen en ppm.

Resultado Esperado: res Esperado.ppm → imagen en ppm.

Salida por pantalla:

```
Introduzca uno de los siguientes comandos disponibles:  
1. comprimir  
2. descomprimir  
3. comprimirYdescomprimir  
4. cambiarAlgoritmoTextoPredeterminado  
5. salir  
3  
Escriba el path relativo del fichero a comprimir (.txt o .ppm!)  
JuegosDePruebas/Prueba7/cayuga 1.ppm  
Indique la calidad de compresión a usar (del 1 al 7)  
4  
Se inicia el proceso  
El proceso ha tardado 2.24223618s. El cambio de tamaño pasa de 1440060B a 723217B con diferencia de 716843B que resulta en un 50.0% del archivo original.  
El proceso ha tardado 1.690824598s. El cambio de tamaño pasa de 723217B a 1440015B con diferencia de 716798B que resulta en un 199.0% del archivo original.  
El archivo /home/jan/Documents/PROP/PROPCompressor/EXE/CtrlProcesos/JuegosDePruebas/Prueba7/cayuga 1.ppm se ha comprimido y descomprimido correctamente!
```

Salida: cayuga_1_out.ppm → tiene el mismo contenido que el resultado esperado

Veredicto: OK

Prueba 8

Descripción: Probar el método de cambio de algoritmo predeterminado que por código es el lzss.

Objetivo: Comprobar que se realiza la compresión de un archivo .txt usando el algoritmo .lz78, después de cambiar el algoritmo predeterminado de lzss a lz78.

Driver Usado: CtrlProcesosDriver

Entrada: el_quijote.txt → fichero que contiene 1.060.259 bytes bytes que son caracteres ASCII y ASCIIextended.

Resultado Esperado: res Esperado.lz78 → El resultado esperado de la compresión es un archivo .lz78 que contiene el resultado de comprimir el_quijote.txt ya que LZ78 es el algoritmo por defecto inicial.

Salida por pantalla:

Introduzca uno de los siguientes comandos disponibles:

1. comprimir
2. descomprimir
3. comprimirYdescomprimir
4. cambiarAlgoritmoTextoPredeterminado
5. salir

4
El algoritmo de compresión de texto predeterminado es LZSS.

Escriba el algoritmo de compresión que quiera usar de manera predeterminada, de entre los siguientes:

lzss
lz78
lzw
lz78

Algoritmo de texto predeterminado cambiado a LZ78.

Introduzca uno de los siguientes comandos disponibles:

1. comprimir
2. descomprimir
3. comprimirYdescomprimir
4. cambiarAlgoritmoTextoPredeterminado
5. salir

1

Escriba el path relativo del fichero a comprimir (.txt o .ppm!)

JuegosDePruebas/Prueba8/el_quijote.txt

Escriba el algoritmo de compresión que quiera usar, de entre los siguientes:

predeterminado
lzss
lz78
lzw
predeterminado

Se inicia el proceso

El proceso ha tardado 0.448932531s. El cambio de tamaño pasa de 1060259B a 571982B con diferencia de 488277B que resulta en un 53.0% del archivo original.

El archivo /home/jan/Documents/PROP/PROPCompressor/EXE/CtrlProcesos/JuegosDePruebas/Prueba8/el_quijote.txt se ha comprimido correctamente!

Salida: el_quijote.lz78 → tiene el mismo contenido que el resultado esperado

Veredicto: OK

DatosProceso

Prueba 1

Descripción: Probar el método getDiffSize en compresión de la clase DatosProceso.

Objetivo: Comprobar que hace el cálculo correcto de diferencia de tamaño del proceso.

Driver Usado: DatosProcesoDriver

Entrada: todos los parámetros requeridos en el driver, poniendo énfasis en oldSize y newSize.

Resultado Esperado: creación de un objeto DatosProceso, el cual hace el cálculo de getDiffSize en compresión correctamente.

Salida por pantalla:

```
Bienvenido al driver para DatosProceso

Escoja una opción:
1. Crear una instancia de DatosProceso
2. Salir
1
Introduzca a continuación un time (por ejemplo 123):
1234
Introduzca a continuación un oldSize (por ejemplo 1000):
1000
Introduzca a continuación un newSize (por ejemplo 500):
564
Introduzca a continuación esCompresion, si es compresión--> '1' o si no lo es--> '0'
1
A continuación se creará una instancia de DatosProceso con time :1234 oldsize: 1000 newSize: 564 y boolean esCompresion true
El proceso ha tardado 1.234E-6s. El cambio de tamaño pasa de 1000B a 564B con diferencia de 436B que resulta en un 56.0% del tamaño original.
Ahora se probará el getDiffSize:
El resultado de DiffSize es: 436
Ahora se probará el getDiffSizePercentage:
El resultado de getDiffSizePercentage es: 56.0
Ahora se probará el getTiempo:
El resultado de getTiempo es: 1234
Ahora se probará el getNewSize:
El resultado de getNewSize es: 564
Ahora se probará el getOldSize:
El resultado de getOldSize es: 1000
```

Salida:

Veredicto: OK

Prueba 2

Descripción: Probar el método getDiffSize en descompresión de la clase DatosProceso.

Objetivo: Comprobar que hace el cálculo correcto de diferencia de tamaño del proceso.

Driver Usado: DatosProcesoDriver

Entrada: todos los parámetros requeridos en el driver, poniendo énfasis en oldSize y newSize.

Resultado Esperado: creación de un objeto DatosProceso, el cual hace el cálculo de getDiffSize en descompresión correctamente.

Salida por pantalla:

```
Bienvenido al driver para DatosProceso
```

```
Escoja una opción:
```

1. Crear una instancia de DatosProceso
2. Salir

```
1
```

```
Introduzca a continuación un time (por ejemplo 123):
```

```
123
```

```
Introduzca a continuación un oldSize (por ejemplo 1000):
```

```
500
```

```
Introduzca a continuación un newSize (por ejemplo 500):
```

```
1000
```

```
Introduzca a continuación esCompresion, si es compresión--> '1' o si no lo es--> '0'
```

```
0
```

```
A continuación se creará una instancia de DatosProceso con time :123 oldsize: 500 newSize: 1000 y boolean esCompresion false  
El proceso ha tardado 1.23E-7s. El cambio de tamaño pasa de 500B a 1000B con diferencia de 500B que resulta en un 200.0% del tamaño original.
```

```
Ahora se probará el getDiffSize:
```

```
El resultado de DiffSize es: 500
```

```
Ahora se probará el getDiffSizePercentage:
```

```
El resultado de getDiffSizePercentage es: 200.0
```

```
Ahora se probará el getTiempo:
```

```
El resultado de getTiempo es: 123
```

```
Ahora se probará el getNewSize:
```

```
El resultado de getNewSize es: 1000
```

```
Ahora se probará el getOldSize:
```

```
El resultado de getOldSize es: 500
```

Salida:

Veredicto: OK

Prueba 3

Descripción: Probar el método getDiffSizePercentage en compresión de la clase DatosProceso.

Objetivo: Comprobar que hace el cálculo correcto de diferencia de porcentaje del proceso.

Driver Usado: DatosProcesoDriver

Entrada: todos los parámetros requeridos en el driver, poniendo énfasis en oldSize y newSize.

Resultado Esperado: creación de un objeto DatosProceso, el cual hace el cálculo de getDiffSizePercentage en compresión correctamente.

Salida por pantalla:

```
Bienvenido al driver para DatosProceso
```

```
Escoja una opción:
```

1. Crear una instancia de DatosProceso
2. Salir

```
1
```

```
Introduzca a continuación un time (por ejemplo 123):
```

```
1234
```

```
Introduzca a continuación un oldSize (por ejemplo 1000):
```

```
1000
```

```
Introduzca a continuación un newSize (por ejemplo 500):
```

```
564
```

```
Introduzca a continuación esCompresion, si es compresión--> '1' o si no lo es--> '0'
```

```
1
```

```
A continuación se creará una instancia de DatosProceso con time :1234 oldsize: 1000 newSize: 564 y boolean esCompresion true
```

```
El proceso ha tardado 1.234E-6s. El cambio de tamaño pasa de 1000B a 564B con diferencia de 436B que resulta en un 56.0% del tamaño original.
```

```
Ahora se probará el getDiffSize:
```

```
El resultado de DiffSize es: 436
```

```
Ahora se probará el getDiffSizePercentage:
```

```
El resultado de getDiffSizePercentage es: 56.0
```

```
Ahora se probará el getTiempo:
```

```
El resultado de getTiempo es: 1234
```

```
Ahora se probará el getNewSize:
```

```
El resultado de getNewSize es: 564
```

```
Ahora se probará el getOldSize:
```

```
El resultado de getOldSize es: 1000
```

Salida:

Veredicto: OK

Prueba 4

Descripción: Probar el método getDiffSizePercentage en descompresión de la clase DatosProceso.

Objetivo: Comprobar que hace el cálculo correcto de diferencia de porcentaje del proceso.

Driver Usado: DatosProcesoDriver

Entrada: todos los parámetros requeridos en el driver, poniendo énfasis en oldSize y newSize.

Resultado Esperado: creación de un objeto DatosProceso, el cual hace el cálculo de getDiffSizePercentage en descompresión correctamente.

Salida por pantalla:

```
Bienvenido al driver para DatosProceso

Escoja una opción:
1. Crear una instancia de DatosProceso
2. Salir
1
Introduzca a continuación un time (por ejemplo 123):
123
Introduzca a continuación un oldSize (por ejemplo 1000):
500
Introduzca a continuación un newSize (por ejemplo 500):
1000
Introduzca a continuación esCompresion, si es compresión--> '1' o si no lo es--> '0'
0
A continuación se creará una instancia de DatosProceso con time :123 oldsize: 500 newSize: 1000 y boolean esCompresion false
El proceso ha tardado 1.23E-7s. El cambio de tamaño pasa de 500B a 1000B con diferencia de 500B que resulta en un 200.0% del tamaño original.
Ahora se probará el getDiffSize:
El resultado de DiffSize es: 500
Ahora se probará el getDiffSizePercentage:
El resultado de getDiffSizePercentage es: 200.0
Ahora se probará el getTiempo:
El resultado de getTiempo es: 123
Ahora se probará el getNewSize:
El resultado de getNewSize es: 1000
Ahora se probará el getOldSize:
El resultado de getOldSize es: 500
```

Salida:

Veredicto: OK

Prueba 5

Descripción: Probar el método getTiempo de la clase DatosProceso.

Objetivo: Comprobar que devuelve el tiempo correctamente del proceso.

Driver Usado: DatosProcesoDriver

Entrada: todos los parámetros requeridos en el driver, poniendo énfasis en time.

Resultado Esperado: creación de un objeto DatosProceso, el cual hace el cálculo de getTiempo correctamente.

Salida por pantalla:

```

Escoga una opción:
1. Crear una instancia de DatosProceso
2. Salir
1
Introduzca a continuación un time (por ejemplo 123):
123
Introduzca a continuación un oldSize (por ejemplo 1000):
1000
Introduzca a continuación un newSize (por ejemplo 500):
500
Introduzca a continuación esCompresion, si es compresión--> '1' o si no lo es--> '0'
1
A continuación se creará una instancia de DatosProceso con time :123 oldsize: 1000 newSize: 500 y boolean esCompresion true
El proceso ha tardado 1.23E-7s. El cambio de tamaño pasa de 1000B a 500B con diferencia de 500B que resulta en un 50.0% del tamaño original.
Ahora se probará el getDiffSize:
El resultado de DiffSize es: 500
Ahora se probará el getDiffSizePercentage:
El resultado de getDiffSizePercentage es: 50.0
Ahora se probará el getTiempo:
El resultado de getTiempo es: 123
Ahora se probará el getNewSize:
El resultado de getNewSize es: 500
Ahora se probará el getOldSize:
El resultado de getOldSize es: 1000
Ahora se probará el isSatisfactorio:
El resultado de isSatisfactorio es: satisfactorio

```

Salida:

Veredicto: OK

Prueba 6

Descripción: Probar el método getOldSize de la clase DatosProceso.

Objetivo: Comprobar que devuelve el tamaño del archivo original del proceso.

Driver Usado: DatosProcesoDriver

Entrada: todos los parámetros requeridos en el driver, poniendo énfasis en oldSize.

Resultado Esperado: creación de un objeto DatosProceso, el cual obtiene con getOldSize el oldSize correctamente.

Salida por pantalla:

```

Escoga una opción:
1. Crear una instancia de DatosProceso
2. Salir
1
Introduzca a continuación un time (por ejemplo 123):
123
Introduzca a continuación un oldSize (por ejemplo 1000):
1000
Introduzca a continuación un newSize (por ejemplo 500):
500
Introduzca a continuación esCompresion, si es compresión--> '1' o si no lo es--> '0'
1
A continuación se creará una instancia de DatosProceso con time :123 oldsize: 1000 newSize: 500 y boolean esCompresion true
El proceso ha tardado 1.23E-7s. El cambio de tamaño pasa de 1000B a 500B con diferencia de 500B que resulta en un 50.0% del tamaño original.
Ahora se probará el getDiffSize:
El resultado de DiffSize es: 500
Ahora se probará el getDiffSizePercentage:
El resultado de getDiffSizePercentage es: 50.0
Ahora se probará el getTiempo:
El resultado de getTiempo es: 123
Ahora se probará el getNewSize:
El resultado de getNewSize es: 500
Ahora se probará el getOldSize:
El resultado de getOldSize es: 1000
Ahora se probará el isSatisfactorio:
El resultado de isSatisfactorio es: satisfactorio

```

Salida:

Veredicto: OK

Prueba 7

Descripción: Probar el método getNewSize de la clase DatosProceso.

Objetivo: Comprobar que devuelve el tamaño del archivo nuevo del proceso.

Driver Usado: DatosProcesoDriver

Entrada: todos los parámetros requeridos en el driver, poniendo énfasis en newSize.

Resultado Esperado: creación de un objeto DatosProceso, el cual obtiene con getNewSize el newSize correctamente.

Salida por pantalla:

```
Escoja una opción:  
1. Crear una instancia de DatosProceso  
2. Salir  
1  
Introduzca a continuación un time (por ejemplo 123):  
123  
Introduzca a continuación un oldSize (por ejemplo 1000):  
1000  
Introduzca a continuación un newSize (por ejemplo 500):  
500  
Introduzca a continuación esCompresion, si es compresión--> '1' o si no lo es--> '0'  
1  
A continuación se creará una instancia de DatosProceso con time :123 oldsize: 1000 newSize: 500 y boolean esCompresion true  
El proceso ha tardado 1.23E-7s. El cambio de tamaño pasa de 1000B a 500B con diferencia de 500B que resulta en un 50.0% del tamaño original.  
Ahora se probará el getDiffSize:  
El resultado de DiffSize es: 500  
Ahora se probará el getDiffSizePercentage:  
El resultado de getDiffSizePercentage es: 50.0  
Ahora se probará el getTiempo:  
El resultado de getTiempo es: 123  
Ahora se probará el getNewSize:  
El resultado de getNewSize es: 500  
Ahora se probará el getOldSize:  
El resultado de getOldSize es: 1000  
Ahora se probará el isSatisfactorio:  
El resultado de isSatisfactorio es: satisfactorio
```

Salida:

Veredicto: OK

Prueba 8

Descripción: Probar el método isSatisfactorio de la clase DatosProceso.

Objetivo: Comprobar que devuelve si el proceso ha sido satisfactorio.

Driver Usado: DatosProcesoDriver

Entrada: todos los parámetros requeridos en el driver, poniendo énfasis en que diffSize sea positivo, y que por tanto, compresión o descompresión han llevado a cabo su cometido.

Resultado Esperado: creación de un objeto DatosProceso, el cual obtiene si el proceso ha sido satisfactorio o no.

Salida por pantalla:

```
Escoja una opción:  
1. Crear una instancia de DatosProceso  
2. Salir  
1  
Introduzca a continuación un time (por ejemplo 123):  
123  
Introduzca a continuación un oldSize (por ejemplo 1000):  
1000  
Introduzca a continuación un newSize (por ejemplo 500):  
500  
Introduzca a continuación esCompresion, si es compresión--> '1' o si no lo es--> '0'  
1  
A continuación se creará una instancia de DatosProceso con time :123 oldsize: 1000 newSize: 500 y boolean esCompresion true  
El proceso ha tardado 1.23E-7s. El cambio de tamaño pasa de 1000B a 500B con diferencia de 500B que resulta en un 50.0% del tamaño original.  
Ahora se probará el getDiffSize:  
El resultado de DiffSize es: 500  
Ahora se probará el getDiffSizePercentage:  
El resultado de getDiffSizePercentage es: 50.0  
Ahora se probará el getTiempo:  
El resultado de getTiempo es: 123  
Ahora se probará el getNewSize:  
El resultado de getNewSize es: 500  
Ahora se probará el getOldSize:  
El resultado de getOldSize es: 1000  
Ahora se probará el isSatisfactorio:  
El resultado de isSatisfactorio es: satisfactorio
```

Salida:

Veredicto: OK