PROP
Compressor d’arxius

Relació Classes Implementades - Membres

Identificador de Grup: grup-6.4

Joan Bosch Pons
Joan Salvador Magrans
Clara Sánchez Flaquer
Daniela Zhao

Joan Bosch Pons :

- LZ78
- Algorisme
- Resultat Algorisme
- Controlador Fitxer
- Driver LZ78
- Driver Algorisme
- Driver Resultat Algorisme
- Driver Controlador Fitxer

2a Entrega:

- Vista Compresor, pestanya Descomprimir
- Controlador Arxius (mètodes comprimir i descomprimir fitxer)
- Fitxer HTML Ajuda Descomprimir

Joan Salvador Magrans:

- JPEG
- Controlador Dades
- Driver JPEG

2a Entrega:

- Vista Compresor, pestanya Comprimir
- Fitxer HTML Ajuda Comprimir
- Huffman JPEG
- Controlador Algorisme

Clara Sánchez Flaquer:

- LZW
- Fitxer
- Driver Fitxer
- Controlador Carpeta
- Driver LZW
- JUnit Fitxer
- MainJUnit

2a Entrega:

- Controlador Arxius (mètodes descomprimir i comprimir carpeta)
- Vista Compressor, pestanya Estadístiques locals
- Fitxer HTML Ajuda Estadístiques

Daniela Zhao:

- LZSS
- Estadístiques Locals
- Estadístiques Globals
- Controlador Estadístiques Globals
- Driver LZSS
- Driver Estadístiques Locals
- Driver Estadístiques Globals
- Driver Controlador Estadístiques Globals

2a Entrega:

- Vista Compressor, pestanya Comparar
- Vista Comparar
- Vista Ajuda
- Fitxer HTML Ajuda Comparar
- Controlador Presentació
- Main.java

LLISTAT DE CANVIS REALITZATS RESPECTE LA PRIMERA ENTREGA:

1. Explicació de les funcions i atributs del Diagrama de classes *(Línia 20)*
   Al document de *Diagrama_de_classes*, s'han afegit una breu descripció dels atributs i mètodes que contenen les classes.

2. Canvi d'implementació de strings a bytes a cada algorisme *(Línia 23)*
   Com vas comentar a la primera entrega, no acabem d'optimitzar la compressió utilitzant strings, per tant vam decidir canviar la compressió d'strings a una seqüència de bytes. Cadascú va haver de modificar el seu codi en un projecte local a IntelliJ IDEA per adaptar-ho i comprimir/descomprimir amb el tipus byte[].
   Un cop vam tenir una versió funcionant de comprimir/descomprimir una seqüència de bytes, ho vam traslladar al projecte del respositori de gitLab i vam adaptar les classes afectades.

3. Funció get_Instance de la classe Algorisme *(Línia 30)*
   Creadores de cada subclasse d'Algorisme canviada per tal d'utilitzar la clàusula *super* i que aquestes retornin una instància de la pròpia classe en comptes de la clase genèrica *Algorisme*.

4. Canviat el mode automàtic *(Línia 31):*
   A la primera entrega, el mode automàtic que vam implementar, funcionava de la següent manera; si es passava un fitxer amb una extensió .txt, es comprimia el fitxer amb els tres algorismes de compressió de textos. Anàlogament pels arxius .ppm (tot i que en aquest cas només tenim l'algorisme JPEG). Quan tots els algorismes havien finalitzat la seva compressió, es mirava quin dels tres ocupava menys i s'escollia aquell com a algorisme automàtic.
   Centrant-nos ara amb els algorismes de compressió de text, la modificació que hem fet ha estat canviar el mode automàtic fent un estudi previ per veure com actuen els diferents algorismes amb fitxers de diferents mides.

En funció del grau de compressió i del temps de compressió, hem decidit que l'algorisme LZSS és el més adient per fitxers .txt de 0B a 75kB, ja que amb comparació amb LZW i LZ78 era el que comprimia més ràpid i amb un grau de compressió major. A partir d'aquest valor i per fitxers de més de 75kB, l'algorisme més adient és el LZW, pel mateix raonament anterior.

Igual que a l'anterior entrega, quan es detecta una extensió .ppm directament es comprimeix utilitzant l'algorisme JPEG.

Els resultats de les proves que hem realitzat per arribar a aquestes conclusions son les següents:

Fitxer de 200 kB:

	LZ78	LZSS	**LZW**
Temps compressió (ms)	64	218	**61,14**
Grau compressió (%)	56,34	65,04	**68,07**

Fitxer de 100 kB:

	LZ78	LZSS	**LZW**
Temps compressió (ms)	37	139	**36**
Grau compressió (%)	51,08	64,61	**64,17**

Fitxer de 50 kB:

	LZ78	**LZSS**	LZW
Temps	15,74	**80**	16

compressió (ms)			
Grau compressió (%)	45,3	**63,73**	58,59

Fitxer de 75 kB:

	LZ78	**LZSS**	LZW
Temps compressió (ms)	10	**81,89**	17
Grau compressió (%)	49,48	**64,56**	62,44

Com podem observar a la darrera prova, els algorismes LZSS i LZW obtenen uns resultats molt semblants en quant al grau de compressió però LZSS és millor encara que el temps és pitjor, i és per això que hem decidit aplicar aquest llindar.

5. Afegir comentaris en capçaleres *(Línia 32/35)*

   Cadascú ha afegit en les classes que ha implementat, comentaris a les capçaleres de les funcions indicant què fa aquella funció. En les funcions de les funcionalitats principals s'ha especificat el PRE i POST de cada funció.

6. LZSS, Claredat *(Línia 42)*

   A la primera entrega, a la classe de LZSS només tenia les funcions de *comprimirContingut* i *descomprimirContingut* i per tant, estava tot el codi en una funció i és cert que era una mica difícil de llegir a causa de la quantitat de bucles que hi ha.

   Ho he estat pensant i he pogut extreure un mètode, *obtenirMatchLength* per reduir una mica la quantitat de línies de codi dins de la funció comprimir.

No m'he atrevit a extreure més mètodes ja que en tots els altres bucles modifico el HashMap i com que a Java els paràmetres es passen sempre per valor, passar el HashMap com a paràmetre seria una solució ineficient, per tant, he deixat aquella part del codi sense tocar.

7. Reducció de la repetició de codi i comentaris amb una millor explicació del codi a JPEG *(Línia 50)*

8. L'algorisme JPEG ja inclou el zig-zag i Huffman *(Línia 51)*

9. S'ha arreglat l'algorisme LZ78 que no funcionava correctament amb el Quijote. *(Línia 57)*

10. Creació d'una nova classe Controlador Arxius:
    Hem creat una classe nova anomenada Controlador Arxius que incorpora la classe Controlador Fitxer de l'entrega anterior i els mètodes necessaris per comprimir i descomprimir Carpetes. Inclou també una funció *trobarMidesImatge* que s'utilitza per la Vista Comparar.