

En aquest informe escrit hi ha  
detallada tota la documentació  
respectiva a la pràctica de “El  
Compressor”

# El Compressor

Versió de lliurament 1.3

**Anna Llanza Carmona** –

anna.llanza@est.fib.upc.edu

**Adrian Roman Iglesias** –

adrian.roman@est.fib.upc.edu

**Josep Maria Olivé Fernández** –

josep.maria.olive.fernandez@est.fib.upc.edu

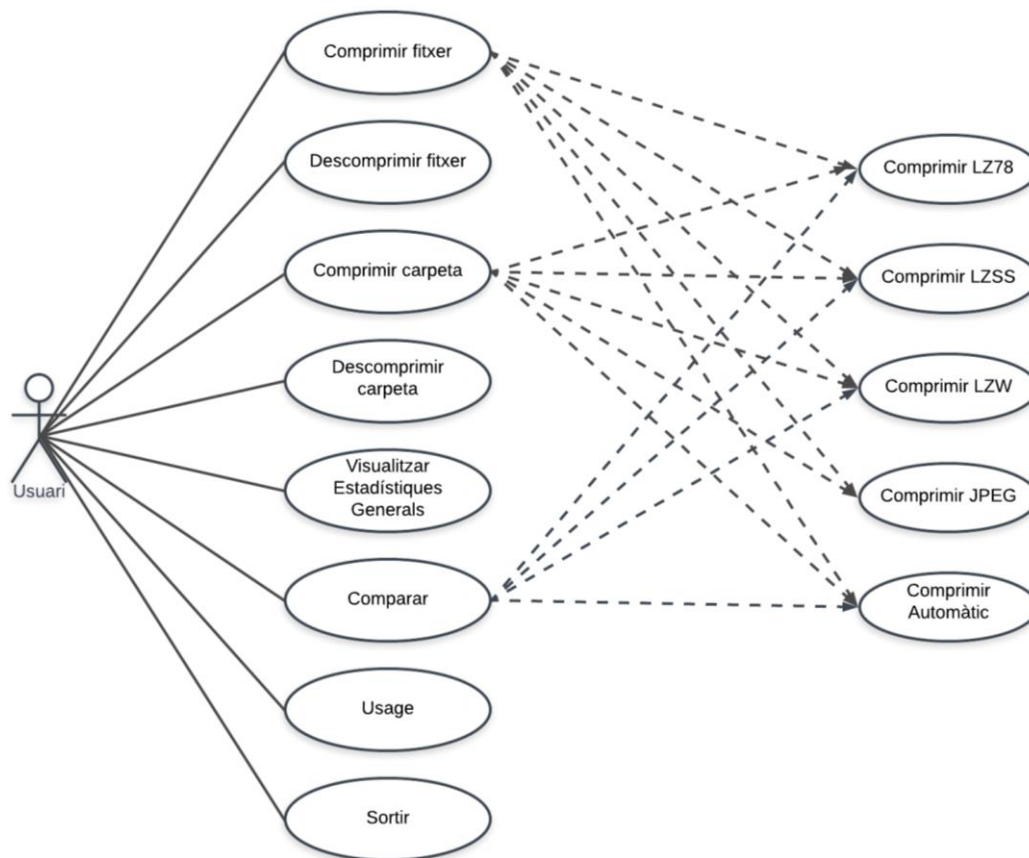
**Roger Gonzalez Herrera** –

roger.gonzalez.herrera@est.fib.upc.edu

## Índex

1. Diagrama dels casos d'ús i corresponent descripció.....pàgina 2
2. Diagrama de classes i descripcions de les classes.....pàgina 7
3. Descripció de les estructures de dades i algorismes de les  
funcionalitats principals.....pàgina 14

## 1. Diagrama dels casos d'ús i corresponent descripció



### Comprimir fitxer:

- Agents: Usuari.
- Descripció: L'usuari introdueix el path d'un fitxer com a paràmetre d'entrada. Aquest fitxer pot ser de tipus text o imatge. Es mostraran les opcions dels algorismes disponibles per la seva compressió i l'usuari en triarà la més adient. Es comprimeix el fitxer i es guarda amb el path anterior afegint l'extensió de l'algorisme utilitzat per comprimir.
- Diàleg: El paràmetre d'entrada és el path del fitxer a comprimir. El paràmetre de sortida són les estadístiques generades pel fitxer introduït. (Grau de compressió, velocitat i temps).
- Errors:
  1. El fitxer no existeix.
  2. El path és incorrecte.

### • Solucions:

1. Ensenyar un missatge informant que el fitxer no existeix.
2. Ensenyar un missatge informant que el path no existeix.

### Descomprimir fitxer:

- Agents: Usuari.
- Descripció: L'usuari introdueix el path d'un fitxer com a paràmetre d'entrada. Aquest fitxer ha hagut d'estar comprimit per un dels nostres algorismes. Es mostrarà l'opció de l'algorisme disponible per la seva descompressió i l'usuari la seleccionarà. Es descomprimeix el fitxer i es guarda amb el path anterior traient l'extensió de l'algorisme.
- Diàleg: El paràmetre d'entrada és el path del fitxer a descomprimir. El paràmetre de sortida són les estadístiques generades pel fitxer introduït (Grau de descompressió, velocitat i temps).
- Errors:
  1. El fitxer no existeix.
  2. El path és incorrecte.
- Solucions:
  1. Ensenyar un missatge informant que el fitxer no existeix.
  2. Ensenyar un missatge informant que el path no existeix.

### Comprimir carpeta:

- Agents: Usuari.
- Descripció: L'usuari introdueix el path d'una carpeta com a paràmetre d'entrada. Aquesta carpeta pot contenir una altra carpeta i/o fitxers. Es mostraran les opcions dels algorismes disponibles per la seva compressió i l'usuari en triarà la més adient. Si el fitxer és de tipus text utilitzarà l'opció triada; per altra banda si el fitxer és de tipus imatge, per defecte, es realitzarà la compressió mitjançant l'algorisme JPEG. Es comprimeix la carpeta fitxer a fitxer i es guarda amb el path anterior afegint l'extensió de l'algorisme.
- Diàleg: El paràmetre d'entrada és el path de la carpeta a comprimir.
- Errors:
  1. Dins la carpeta hi ha un fitxer que no es conegut.
  2. La carpeta no existeix.
  3. El path és incorrecte.
- Solucions:
  1. Ensenyar un missatge informant que la carpeta no és vàlida.
  2. Ensenyar un missatge informant que la carpeta no existeix.
  3. Ensenyar un missatge informant que el path no existeix.

### Descomprimir carpeta:

- Agents: Usuari.
- Descripció: L'usuari introdueix el path d'una carpeta com a paràmetre d'entrada. Aquesta carpeta pot contenir una altra carpeta i/o fitxers comprimits. Es mostrarà l'opció de l'algorisme disponible per la seva descompressió i l'usuari la seleccionarà. Si el fitxer és de tipus text utilitzarà l'opció triada; per altra banda si el fitxer és de tipus imatge, per defecte, es realitzarà la descompressió mitjançant l'algorisme JPEG. Es descomprimeix la carpeta fitxer a fitxer i es guarda amb el path anterior traient l'extensió de l'algorisme.
- Diàleg: El paràmetre d'entrada és el path de la carpeta a descomprimir.
- Errors:
  1. Dins la carpeta hi ha un fitxer que no es conegut.
  2. La carpeta no existeix.
  3. El path és incorrecte.
- Solucions:
  1. Ensenyar un missatge informant que la carpeta no és vàlida.
  2. Ensenyar un missatge informant que la carpeta no existeix.
  3. Ensenyar un missatge informant que el path no existeix.

### Visualitzar estadístiques generals:

- Agents: Usuari.
- Descripció: L'usuari demana veure les estadístiques generals d'un fitxer i d'un algorisme. Es mostrarà les opcions dels algorismes disponibles i l'usuari triarà un. També es necessita que l'usuari indiqui si vol les estadístiques dels fitxers comprimits o descomprimits. Es mostraran els valors estadístics (grau mitjà, velocitat mitjana, temps mitjà).
- Diàleg: El paràmetre d'entrada és el nom de l'algorisme i el tipus de fitxer. El paràmetre de sortida són els valors de les estadístiques generals (grau mitjà, velocitat mitjana, temps mitjà).
- Errors:
  1. No hi ha estadístiques.
- Solucions:
  1. Ensenyar un missatge informant que encara no s'ha comprimit o descomprimit cap tipus d'arxiu.

### Comparar:

- Agents: Usuari.
- Descripció: L'usuari introdueix el path d'un fitxer com a paràmetre d'entrada. Es mostraran les opcions dels algorismes disponibles per la seva compressió i l'usuari en triarà la més adient. Comprimeix el fitxer i el torna a descomprimir. Finalment, mostra els dos fitxers per a visualitzar i comparar.
- Diàleg: El paràmetre d'entrada és el path del fitxer a comprimir.
- Errors:
  1. L'arxiu no existeix.
  2. Ja existeix un arxiu amb el nom de l'arxiu de sortida.
  3. El path és incorrecte.
  4. L'extensió no pertany a ningun dels algorismes utilitzats.
- Solucions:
  1. Ensenyar un missatge informant que l'arxiu no existeix.
  2. Ensenyar un missatge informant que l'arxiu de sortida ja existeix.
  3. Ensenyar un missatge informant que el path no existeix.
  4. Ensenyar un missatge informant que l'extensió no és la correcta.

### Usage:

- Agents: Usuari.
- Descripció: S'obre una finestra amb un missatge indicant com s'utilitza el programa.
- Diàleg: No té paràmetres.
- Errors: Cap.
- Solucions: Cap.

### Sortir:

- Agents: Usuari.
- Descripció: Es tanca el programa.
- Diàleg: No té paràmetres.
- Errors: Cap.
- Solucions: Cap.

### Comprimir LZ78:

- Agents: Usuari.
- Descripció: Es comprimeix el fitxer de text d'entrada amb l'algorisme LZ78.
- Diàleg: L'usuari introdueix un fitxer com a entrada, i es passa com a sortida el contingut del fitxer comprimit i el temps estadístic.
- Errors: Cap.
- Solucions: Cap.

#### Comprimir LZSS:

- Agents: Usuari.
- Descripció: Es comprimeix el fitxer de text d'entrada amb l'algorisme LZSS.
- Diàleg: L'usuari introdueix un fitxer com a entrada, i es passa com a sortida el contingut del fitxer comprimit i el temps estadístic.
- Errors: Cap.
- Solucions: Cap.

#### Comprimir LZW:

- Agents: Usuari.
- Descripció: Es comprimeix el fitxer de text d'entrada amb l'algorisme LZW.
- Diàleg: L'usuari introdueix un fitxer com a entrada, i es passa com a sortida el contingut del fitxer comprimit i el temps estadístic.
- Errors: Cap.
- Solucions: Cap.

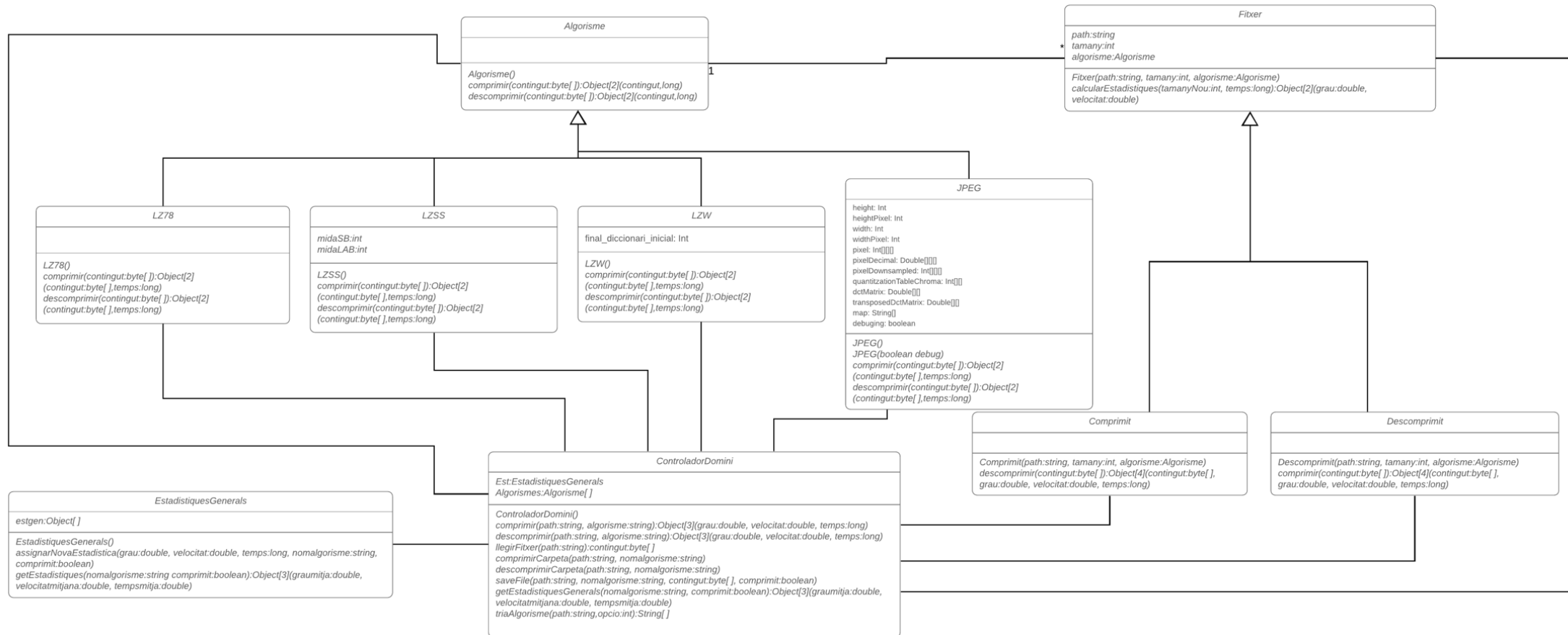
#### Comprimir JPEG:

- Agents: Usuari.
- Descripció: Es comprimeix el fitxer de text d'entrada amb l'algorisme JPEG.
- Diàleg: L'usuari introdueix un fitxer com a entrada, i es passa com a sortida el contingut del fitxer comprimit i el temps estadístic.
- Errors: Cap.
- Solucions: Cap.

#### Comprimir automàtic

- Agents: Usuari.
- Descripció: Es comprimeix el fitxer de text d'entrada amb l'algorisme més eficient per al fitxer.
- Diàleg: L'usuari introdueix un fitxer com a entrada, i es passa com a sortida el contingut del fitxer comprimit i el temps estadístic.
- Errors: Cap.
- Solucions: Cap.

## 2. Diagrama de classes i descripcions de les classes



- Controlador Domini:

- Creadora

Crea un Controlador de Domini.

- triaAlgorisme (path:string) : Algorisme[ ]

Analitza el path d'entrada i delimita els possibles algorismes que pot escollir l'usuari i els retorna.

- llegirFitxer(path:string):byte[]

Crida al controlador de persistència passant-li el path perquè llegeixi aquest fitxer i retorni el contingut del fitxer en un array de bytes.

- comprimir(nom:string, path:string, tamany:string, ext:string, algorisme:Algorisme)

Crida a "llegirFitxer", crea la instància Fitxer de tipus descomprimit amb "crea", També crea l'Algoritme i ordena la navegabilitat entre Fitxer descomprimit i Algoritme amb "assignarAlgorisme", crida la funció "Comprimir" passant-li el contingut, que aquest et retorna el contingut final i totes les estadístiques. Ara tenint contingut i totes les estadístiques, crida el "SaveFile" de la capa de persistència passant-li el path original, nom original, algorisme i el contingut. Posteriorment crida a "AssignarNovaEstadística" per guardar les estadístiques generades.

- descomprimir(nom:string, path:string, tamany:string, ext:string, algorisme:Algorisme)

Crida a "llegirFitxer", crea la instància Fitxer de tipus comprimit amb "crea", També crea l'Algoritme i ordena la navegabilitat entre Fitxer comprimit i Algoritme amb "assignarAlgorisme", crida la funció "Descomprimir" passant-li el contingut, que aquest et retorna el contingut final i totes les estadístiques. Ara tenint contingut i totes les estadístiques, crida el "SaveFile" de la capa de persistència passant-li el path original, nom original, algorisme i el contingut. Posteriorment crida a "AssignarNovaEstadística" per guardar les estadístiques generades.

- comprimircarpeta(nom:string, path:string, tamany:string, ext:string)

- descomprimircarpeta(nom:string, path:string, tamany:string, ext:string)

- saveFile(path:string, nom:string, algoritme:string contingut:byte[])

Modifica el path posant un nou nom amb l'extensió de l'algorisme si es comprimit i si es descomprimit treu el nom de l'algorisme; i crida a la funció "SaveFile" de persistència amb el nou path i el mateix contingut.

- getEstadistiquesGenerals():Tupletype(nom\_algorisme:String, comprimit:boolean)

Crida a la funció de la classe Estadistiques amb el nom de l'algorisme i un bool dependent de si es comprimit o descomprimit

- **Classe Fitxer:**

- Atributs protected:

1. path:string -> guarda el path del fitxer
2. tamany:int -> guarda el tamany del fitxer
3. algorisme:Algorisme -> guarda l'algorisme amb el que es vol descomprimir

- Fitxer(path:string, tamany:double, Algorisme:algorisme)

Creadora de la classe, on els atributs prenen el valor dels paràmetres.

- CalcularEstadistiques(int tamanyNou, long temps)

Rep un tamanyNou amb el tamany de el fitxer en qüestió després de ser comprimit o descomprimit i un long amb el temps en milisegons de la operació comprimir o descomprimir.

Retorna un Object[2] amb el grau% de compressió o descompressió  $(\text{tamanyNou}/\text{tamanyVell}) * 100$  i amb la velocitat, calculada com  $\text{tamanyVell}/\text{temps}$ .

- **Classe Comprimit:**

- Atributs protected:

4. path:string -> guarda el path del fitxer
5. tamany:int -> guarda el tamany del fitxer
6. algorisme:Algorisme -> guarda l'algorisme amb el que es vol descomprimir

- Comprimit(path:string, tamany:double, Algorisme:algorisme)

Crea un fitxer comprimit, on els atributs prenen el valor dels paràmetres.

- descomprimir(contingut:byte[]):Object[4]

Crida a la funció descomprimir de l'algorisme associat passant-li com a paràmetre el contingut. Rebrà un Object[2] amb el contingut comprimit en un array de bytes i també el temps de descompressió. Aleshores cridarà a la funció calcularEstadistiques que té per herència de Fitxer i aquesta li retornarà un Object[2] amb el grau i la velocitat de descompressió d'aquest arxiu. Aquesta funció descomprimit, retorna un Object[4] amb el contingut en un array de bytes, el grau de descompressió, la velocitat de descompressió i el temps de descompressió.

- **Classe Descomprimit:**

- Atributs protected:

1. path:string -> guarda el path del fitxer

2. `tamany:int` -> guarda el tamany del fitxer
3. `algorisme:Algorisme` -> guarda l'algorisme amb el que es vol descomprimir

- `Descomprimit(path:string, tamany:double, Algorisme:algorisme)`

Creadora de la classe, on els atributs prenen el valor dels paràmetres.

- `Comprimir(contingut:byte[]):Object[4]`

Crida a la funció `comprimir` de l'algorisme associat passant-li com a paràmetre el contingut. Rebrà un `Object[2]` amb el contingut comprimit en un array de bytes i també el temps de descompressió. Aleshores cridarà a la funció `calcularEstadistiques` que té per herència de `Fitxer` i aquesta li retornarà un `Object[2]` amb el grau i la velocitat de descompressió d'aquest arxiu. Aquesta funció `descomprimit`, retorna un `Object[4]` amb el contingut en un array de bytes, el grau de descompressió, la velocitat de compressió i el temps emprat en compressió.

- **Classe Algorisme:**

- `comprimir(contingut:byte[]):Object[2]`

Aquesta funció és abstracta i per tant no té implementació. Les subclasses de `Algorisme` la implementaran per tal de comprimir el contingut que es rep com a paràmetre. Finalment retornaran un objecte que contindrà el contingut comprimit i el temps que s'ha requerit per comprimir.

- `descomprimir(contingut:byte[]):Object[2]`

Aquesta funció és abstracta i per tant no té implementació. Les subclasses de `Algorisme` la implementaran per tal de descomprimir el contingut que es rep com a paràmetre. Finalment retornaran un objecte que contindrà el contingut descomprimit i el temps que s'ha requerit per descomprimir.

- **Classe LZ78:**

- Atributs privats:

1. `Est:EstadistiquesGenerals` -> Dona accés a la classe `EstadistiquesGenerals`
2. `Algorismes:Algorisme[]` -> Dona accés a totes les subclasses d'algorisme possibles

- `LZ78`

Aquesta funció crea la classe LZ78 i inicialitza els seus atributs.

- `comprimir(contingut:byte[]):Object[2]`

Aquesta funció comprimeix el contingut que té com a paràmetre, utilitzant l'algorisme de compressió LZ78. Retorna un `Object[2]` amb un array de bytes amb el contingut comprimit i també retorna el temps que ha trigat en comprimir l'arxiu.

- `descomprimir(contingut:byte[]):Object[2]`

Aquesta funció descomprimeix el contingut que té com a paràmetre, utilitzant l'algorisme de descompressió LZ78. Retorna un `Object[2]` amb un array de bytes amb el contingut descomprimit i també retorna el temps que ha trigat en descomprimir l'arxiu.

- **Classe LZSS:**

- Atributs privats:

1. `midaSB:int` -> guarda la mida màxima que pot tenir el Search Buffer
2. `midaLAB:int` -&code> guarda la mida màxima que pot tenir el Lookahead Buffer

- LZSS

Aquesta funció crea la classe LZSS i inicialitza els seus atributs..

- `comprimir(contingut:byte[]):Object[2]`

Aquesta funció comprimeix el contingut que té com a paràmetre, utilitzant l'algorisme de compressió LZSS. Retorna un `Object[2]` amb un array de bytes amb el contingut comprimit i també retorna el temps que ha trigat en comprimir l'arxiu.

- `descomprimir(contingut:byte[]):Object[2]`

Aquesta funció descomprimeix el contingut que té com a paràmetre, utilitzant l'algorisme de descompressió LZSS. Retorna un `Object[2]` amb un array de bytes amb el contingut descomprimit i també retorna el temps que ha trigat en descomprimir l'arxiu.

- **Classe LZW:**

- Atribut privat:

1. `final_diccionari_inicial: Int`-> guarda la mida amb la que s'inicialitza i omple el diccionari de l'algorisme.

- LZW

Aquesta funció es la constructora de la classe.

- `comprimir(contingut:byte[]):Object[2]`

Aquesta funció comprimeix el contingut que té com a paràmetre, utilitzant l'algorisme de compressió LZW. Retorna un `Object[2]` amb un array de bytes amb el contingut comprimit i també retorna el temps en milisegons que ha trigat en comprimir l'arxiu.

- `descomprimir(contingut:byte[]):Object[2]`

Aquesta funció descomprimeix el contingut que té com a paràmetre, utilitzant l'algorisme de descompressió LZW. Retorna un `Object[2]` amb un array de bytes amb el contingut descomprimit i també retorna el temps en milisegons que ha trigat en descomprimir l'arxiu.

#### ○ Classe JPEG:

- Atributs privats:

1. `height` → guarda la alçada en píxels de la imatge original.
2. `heightPixel` → guarda el múltiple de 8 major o igual a `height` més proper.
3. `width` → guarda la amplada en píxels de la imatge original.
4. `widthPixel` → guarda el múltiple de 8 major o igual a `width` més proper.
5. `int[][][] pixel` → guarda els valors enters dels píxels, representats en RGB i en el cas de YCbCr en guarda només la component Y després de fer el downsampling.
6. `double[][][] pixelDecimal` → guarda els valors decimals de les components Cb i Cr dels píxels quan operem amb elles a les etapes de DCT, I-DCT i quantització, desquantització.
7. `int[][][] pixelDownsampled` → guarda els components Cb i Cr dels píxels, un cop hem aplicat el downsampling de (4:2:2).
8. `int quantizationTableChroma[][]` → taula de quantització única que s'usa en la etapa de quantització i desquantització.
9. `double dctMatrix[][]` → matriu utilitzada en les operacions de DCT i I-DCT, correspon a la matriu DCT d'una matriu de 8x8.
10. `double transposedDctMatrix[][]` → és la transposada de la `dctMatrix`, també necessària per les operacions de DCT i I-DCT.
11. `String[] map` → Diccionari usat en la compressió per Huffman, sempre de mida 256, en la seva posició i, conté la traducció en bits escrits en ASCII d'aquell byte p en codi Huffman.
12. `class Node` → subclasse privada que implementa un node d'un arbre binari. S'utilitza per tal de construir l'arbre de Huffman al comprimir.
13. `boolean debugging` → atribut que indica si volem una sortida verbose quan la classe JPEG està comprimint o descomprimint. Per defecte és sempre false, a no ser que s'especifiqui el contrari amb una creadora especial, que és igual que la per defecte `JPEG()`, però passant-li un paràmetre com a cert o fals, que serà assignat a aquest atribut, `JPEG(true)`, o bé `JPEG(false)`.

- `JPEG()`

Aquesta funció crea un nou objecte JPEG per defecte.

- JPEG(boolean debug)

Aquesta funció creadora, ens genera un objecte JPEG amb un output verbose de tot el que va fent, si el argument debug que li passem és true, sinó és igual que JPEG().

- comprimir(contingut:byte[]):Object[2]

Aquesta funció comprimeix el contingut que té com a paràmetre, utilitzant l'algorisme de compressió JPEG. Retorna un Object[2] amb un array de bytes amb el contingut comprimit i el temps que ha trigat en comprimir l'arxiu.

- descomprimir(contingut:byte[]):Object[2]

Aquesta funció descomprimeix el contingut que té com a paràmetre, utilitzant l'algorisme de descompressió JPEG. Retorna un Object[2] amb un array de bytes amb el contingut descomprimit i el temps que ha trigat en descomprimir l'arxiu.

- **Classe Estadística:**

- Atribut privat:

1. estgen:Object[ ] -> guarda les estadístiques generals, per cada fitxer comprimit o descomprimit i per cada algorisme, és a dir, guarda el grau mitjà, la velocitat mitjana, el temps mitjà i també el numero d'elements per cadascun d'aquests.

- EstadistiquesGenerals()

Crea una instància d'aquesta classe amb l'atribut estgen amb les mitjanes de cada algorisme i per un fitxer comprimit i descomprimit.

- assignarNovaEstadistica(grau:double, velocitat:double, temps:long, nomalgorisme:string, comprimit:boolean)

Afegeix a la mitjana de les dades estadístiques els valors passats com a paràmetre. Per tant, aquesta funció recalcula la mitjana del grau, velocitat i del temps pel fitxer comprimit en el cas de que el paràmetre comprimit sigui true o pel fitxer descomprimit en cas contrari i per l'algorisme corresponent a nomalgorisme.

- getEstadistiques(nomalgorisme:string, comprimit:boolean):Object[3]

Retorna un Object[3] amb el valor del grau mitjà, velocitat mitjana i temps mitjà del fitxer comprimit en el cas de que el paràmetre comprimit sigui true o del fitxer descomprimit en cas contrari i de l'algorisme corresponent a nomalgorisme.

### **Classe Algorisme:**

En aquesta classe hem utilitzat la herència i els mètodes abstractes per tal d'obligar a totes les subclasses a implementar les funcions de comprimir i descomprimir.

### 3. Descripció de les estructures de dades i algorismes de les funcionalitats principals

#### Classe LZ78:

L'algorisme LZ78 es un algorisme de compressió i descompressió que no perd informació.

La seva funció es comprimir i descomprimir un arxiu de tipus text, es a dir, ser capaç de a partir d'un arxiu generar-ne un altre que ocupi menys espai. I, evidentment, fer el pas contrari; a partir del que ocupa menys espai poder generar l'original, es a dir descomprimir.

El procés que he fet servir per crear l'algorisme consta de 3 parts. La primera part es codificar el fitxer obtenint coincidències, la segona part es passar aquestes coincidències a cadena de bytes. La tercera es passar la cadena de bytes al fitxer original. Com podem veure les dues primeres son part de la compressió i la tercera de la descompressió.

Per la primera fem servir dos arrays, un de enters i l'altre de caràcters i un HashMap de Key = String i de Value = enter. El procés consta de, per a cada valor veure si està al diccionari, si no hi es afegir-lo i si no hi es llegir un caràcter mes i tornar a fer el procés. A l'hora d'afegir-lo al diccionari tenim un comptador que cada vegada que afegim una entrada al diccionari va augmentant d'un en un. Tenint així apuntadors diferents. Cada vegada que afegim, a més guardem als arrays l'últim caràcter de la paraula que no ha mostrat coincidència i el valor al qual apunta la ultima coincidència al diccionari. Acabada aquesta fase tenim dos vectors que son la part a codificar en bytes. A més tenim la mida de l'índex mes gran. Si es mes gran a 256 necessitarem 2 bytes per codificar-lo, si es mes gran a  $256^2$  3 i així successivament.

La segona part tracta d'agafar els dos arrays i l'índex mes gran i posar juntament el valor enter en la mida de índex bytes i un altre byte pel char. D'aquesta manera obtenim un array de bytes de mida  $mida\_vector * (index\_mes\_gran + 1)$ . Cal fixar-se en que a vegades el pas de UTF-8 i ASCII pot generar valors que després son interpretats malament i això es un procediment difícil d'obtenir i que pot generar problemes.

Acabat el compilar, ja només queda el descompilar, es a dir la tercera fase. Aquesta consta de crear un HashMap que ara tindrà de Key un string i de valor un enter. Es tracta de recórrer l'array de Bytes i per a cada vegada que l'índex sigui 0 sumar-lo a un String resultat i a mes crear l'entrada al Diccionari, d'altra banda si l'índex no es 0 buscar-lo al Diccionari ja que segur que hi serà perquè s'ha creat anteriorment perquè es el procés revers de la primera fase. Per tant aquí obtenim un String que una vegada acabada la seqüència el transformem a una cadena de bytes i ja obtenim el resultat.

### Classe LZSS:

L'algorisme LZSS és un algorisme de compressió i descompressió de text de tipus lossless. Això vol dir, que a l'hora de fer la compressió i descompressió, no perd informació.

Aquest algorisme aconsegueix que l'arxiu pugui ocupar menys espai. Per a aconseguir-ho, l'algorisme LZSS requereix un Search Buffer i un Lookahead Buffer, els quals són diccionaris que contenen caràcters. El Search Buffer, conté els caràcters que ja han estat comprimits i en canvi, el Lookahead Buffer conté els caràcters que s'han de comprimir.

Per a codificar un caràcter, s'utilitza l'offset i la length. L'offset indica la distància en la que es troba el primer caràcter del Lookahead Buffer amb el caràcter coincident dins del Search Buffer. La length indica el nombre de caràcters seguits coincidents a dins del Search Buffer. A més, també s'utilitza un flag, el qual indica si el caràcter està codificat o simplement és el mateix caràcter que el del fitxer original.

Per tant, a l'hora de comprimir, el procés que és dur a terme fins que s'arriba al final de l'arxiu, és a dir, fins que el Lookahead Buffer està buit, consisteix en agafar el primer caràcter del Lookahead Buffer i mirar si a dins del Search Buffer coincideix amb algun dels caràcters que conté.

Si no coincideix, per a codificar-lo s'escriu flag = 0 i el caràcter sense cap modificació. Si coincideix, comparem el següent caràcter del Lookahead Buffer amb el següent del caràcter coincident dins del Search Buffer i així fins que deixi de coincidir o s'arribi al final del Search Buffer o al final del Lookahead Buffer. En el cas en què el nombre de caràcters coincidents sigui major que 2, aleshores per a codificar-ho, s'escriu flag = 1 i un caràcter que està compost per l'offset i la length (que ha de ser major que 2, però més petit que l'offset). Per altra banda, si la length és menor que 3, és preferible codificar-ho amb flag = 0 i el caràcter original.

Una vegada s'ha codificat un caràcter o un conjunt de caràcters, s'ha d'afegir els caràcters accedits, a dins del Search Buffer i treure'ls del Lookahead Buffer per a que en la codificació del següent caràcter es pugui tindre en compte els últims codificats. Si el flag és 0, només s'haurà d'afegir i treure un sol caràcter, però en el cas del flag igual a 1, això es fa amb tots els caràcters codificats.

Després d'afegir el caràcter o el conjunt de caràcters a dins del Search Buffer, s'ha de comprovar que aquest no tingui un nombre de caràcters més gran que 4096. Si és així, s'hauran de treure del principi del Search Buffer el nombre de caràcters suficients fins que la mida torni a ser 4096. A més, també està limitat el valor de la length, ja que aquesta només pot ser de 18 caràcters coincidents.

Aquestes limitacions es necessiten per a així poder codificar l'offset i la length en un sol caràcter. Per això s'utilitzen 12 bits per a l'offset (el valor màxim és 4096) i 4 bits per a la length (el valor màxim és 16 + 2 del mínim de caràcters coincidents).

A partir d'aquí, l'algorisme de compressió no acaba fins que el Lookahead Buffer estigui buit i per tant es tornarà a repetir el procés fins que sigui així.

El procés de descompressió, és més senzill que el de compressió, ja que depèn del valor del flag. Si aquest és igual a 0, el caràcter descomprimit és el mateix que el que llegim. Aleshores aquest s'haurà d'afegir al Search Buffer i treure del Lookahead Buffer.

Si el flag és igual a 1, això voldrà dir que el caràcter està codificat i per tant s'haurà de descompondre en offset i length. Quan obtenim aquests dos valors, ens posem a una distància igual a la de l'offset a dins del Search Buffer i la length indicarà el conjunt de caràcters que s'han de descomprimir. Aquests també s'hauran d'afegir al Search Buffer i treure del Lookahead Buffer.

En aquest algorisme de descompressió, s'ha de tenir en compte els límits del tamany del Lookahead Buffer (18) i del Search Buffer (4096). A més no s'acaba fins que l'arxiu a descomprimir estigui buit.

### Classe LZW:

L'algorisme LZW es un algorisme de compressió i descompressió de arxius de text sense pèrdua. I per tant al comprimir i descomprimir qualsevol arxiu de text no contindrà cap pèrdua.

Per poder **comprimir**, utilitza un diccionari en el qual ja hi ha tots els caràcters que pot llegir. Per tant aquest diccionari s'ha d'omplir amb tots els caràcters possibles (ascii o fins on considerem per tal d'abastar mes o menys tipus de caràcters) amb la seva posició. Un cop s'ha inicialitzat el diccionari l'algorisme llegeix un caràcter de l'entrada i el concatena a una variable buida(string). Després va llegint caràcter a caràcter de l'entrada, i per cada caràcter llegit mira si al diccionari existeix la variable(string) + el nou caràcter, si és així la variable(string) se li afegeix el nou caràcter i de nou torna a llegir.

En canvi si entrada (la variable(string) + el nou caràcter) no estan al diccionari agafa el codi del diccionari associat a la entrada variable(string) (El codi és la posició de la entrada del diccionari) i la afegeix a una llista de enters que serà la sortida. També, afegeix al diccionari la variable(string) + el nou caràcter, amb el codi de la posició que tindrà. I per últim la variable(string) es igual a el nou caràcter.

Així fins que no quedin més caràcters per llegir.

Per implementar el diccionari de comprimir utilitza un hashmap ja que permet buscar si existeix alguna entrada de manera eficient.

I per afegir el codi de cada entrada s'utilitza la variable global final diccionari inicial per tal de veure fins on està omplert el diccionari inicialment. I llavors anar augmentant una variable entera (final\_local) la qual està inicialitzada amb final\_diccionari\_inicial.

Després de fer tot el procés de compressió per tal de retornar la llista de sortida en un array de bytes es mira cada enter de la sortida i es converteix en 2 bytes i s'afegeix a l'array de bytes, i en el cas de que no hi capigui en 2 bytes s'afegeixen 2 bytes amb tot 0 a l'array de bytes i després s'afegeix l'enter convertit en 3 bytes el qual té un límit de  $2^{24} - 1 = 16\,777\,215$ . També es mesura el temps que tarda el procés i retorna un Object[] amb l'array de bytes i el temps.

Per poder **descomprimir** rep un array de bytes el qual es converteix a un d'enters (tenint en compte la codificació que a la funció comprimir li hem donat, és a dir cada 2 bytes un int i si els 2 bytes són 0 llegir un enter en 3 bytes). S'inicialitza un diccionari amb els final\_diccionari\_inicial primers caràcters (ascii o més com es vulgui) però sempre inicialitzant amb els mateixos que s'ha fet a comprimir.

Un cop inicialitzat el diccionari, s'agafa el primer valor de l'entrada i es guarda a la variable vell i s'agafa el respectiu caràcter del diccionari i s'afegeix a la sortida. També es s'afegeix aquest caràcter a una variable(string) i a una variable(caràcter)

Després comença el procés de descompressió, s'agafa el següent enter de l'entrada i es mira si l'enter és més gran que el diccionari.

Si es així, significa que no tenim el seu string associat al nostre diccionari, llavors fem que `variable(string)` sigui igual a `entrada[vell]` i afegim la `variable(caràcter)` a la `variable(string)`.

Si no es així, vol dir que tenim la entrada al nostre diccionari i per fem que la `variable(string)` tingui el string associat del diccionari a la posició que ens marca l'enter de la entrada.

I finalment estigui al diccionari o no, safegeix la `variable(s)` a la sortida, la `variable caràcter` s'igual a el primer caràcter de la `variable(string)`, s'afegeix al diccionari una nova entrada (`string_asociat_(entrada[vell]+variable(caràcter))`). I es fa que el enter vell sigui igual al de la entrada.

I es torna a començar el procés llegint el següent enter de la entrada.

Per implementar el diccionari aquest cop hem utilitzat una llista de strings i no en un map ja que en tot moment sabem la posició de cada string i per saber si hi es o no només cal mirar la llargada de la llista.

Per últim es retorna la sortida convertida en bytes automàticament, i el temps que ha trigat el procés en un `Object[]`.

### Classe JPEG:

L'algoritme JPEG que hem programat és un dels molts estàndards que trobem al llarg dels anys per aquest mètode, que va fer la seva primera aparició al 1992. A continuació, detallarem pas per pas les tècniques que utilitza el JPEG per aconseguir el seu objectiu principal, reduir l'espai d'una imatge reduint la seva qualitat amb el menor impacte visual possible.

En primer lloc, llegim un arxiu .ppm, que conté les dimensions de la imatge, i el valor de tots els seus píxels en RGB. Guardem aquests valors en una matriu de  $n \times m$ , on  $n$  i  $m$  són la alçada i amplada de la imatge, respectivament. Aquesta matriu, a més a més, té profunditat tres, una subcapa per cada component de RGB.

En segon lloc, modifiquem les dimensions de la matriu esmentada anteriorment, per tal de fer  $n$  i  $m$  múltiples de vuit, i així poder dividir els píxels de la imatge en grups de  $8 \times 8$  píxels.

En tercer lloc, fem la transformació corresponent de RGB a YCbCr, que ens representa la imatge amb la seva component Luma (Y), la llum, i amb Cb i Cr que són la diferència entre el color blau i Luma, i la diferència entre el color vermell i Luma, respectivament.

És important fer aquesta transformació, ja que l'ull humà és molt més sensible als canvis en la llum d'una imatge que no als colors, això ens permetrà, a continuació, reduir la quantitat de color de la imatge deixant la component Luma intacta.

En quart lloc, centrem tots els valors de Cb i Cr en 0, restant 128. Això ens permetrà aplicar la transformació DCT en el sisè pas, ja que aquesta funció, basada en el sinus, treballa amb valors d'entre -1 i 1.

En cinquè lloc, apliquem un downsampling de (4:2:2), és a dir, reduïm a la meitat la quantitat de color de la imatge tant vertical com horitzontalment. És important remarcar, que la component Luma no es modifica.

D'entre totes les opcions, hem triat la opció (4:2:2), ja que és la que més quantitat de color redueix, i per tant, potencialment més espai redueix, però al no tocar la component luminica, la qualitat no es veu afectada negativament.

En sisè lloc, apliquem la transformació DCT2 a Cb i Cr. Hi ha, altrament, diferents mètodes per aplicar aquesta transformació, personalment hem preferit el que utilitza les matrius fixes  $T$  i  $T$  transposada i les multiplica pel grup de  $8 \times 8$  components de Cb o Cr. Aquest mètode, en quant a qualitat de imatge, ens ha resultat molt més efectiu que no altres que hem trobat. A més a més, la seva implementació és molt més neta, ja que al tenir blocs fixos de  $8 \times 8$ , les matrius  $T$  i  $T$  transposada sempre seran les mateixes.

En setè lloc, quantitzem, que consisteix en dividir cada bloc de  $8 \times 8$  de Cb i Cr entre una matriu de  $8 \times 8$ , cada posició entre la mateixa posició de l'altra matriu. I a continuació arrodonim i guardem el resultat. Això ens serveix per arrodonir a zero tots els valors petits que ens ha deixat la transformació DCT.

Finalment, utilitzem la compressió per Huffman, que assigna una representació amb menor quantitat de bits als bytes que més es repeteixen. Fem la traducció de la cadena de bytes que obtenim al llegir Y, Cb i Cr per blocs i cada bloc en zig-zag, per tal d'acumular el major nombre de zeros al final, i els emmagatzemem a una nova cadena de bytes, amb el diccionari corresponent per tal de poder tornar-los al seu estat original.

Per tal de desfer això, i descomprimir la imatge, haurem de executar exactament les mateixes passes a la inversa:

En primer lloc, utilitzem el diccionari emmagatzemat a la cadena de bytes que ens entra per tal de desfer la compressió per Huffman que té aquesta cadena. I a continuació recreem la matriu de píxels amb els components Y, Cb i Cr que conté, la cadena de bytes s'ha de llegir per blocs i per cada bloc, s'ha de desfer la codificació en zig-zag que se li havia aplicat anteriorment.

En segon lloc, multipliquem les components Cb i Cr, en grups de 8x8 píxels, per la taula de quantització utilitzada en la compressió.

En tercer lloc, fem la transformació inversa de DCT2, que implica multiplicar la matriu T transposada, per cada bloc de 8x8 de Cb o Cr i finalment multiplicar el resultat per la matriu T.

En quart lloc, propaguem els valors de Cb i Cr als seus valors adjunts per la part inferior i per la dreta, per tal de desfer el downsampling de (4:2:2), i per tant tornar a tenir (4:4:4).

En cinquè lloc, sumem a tots els valors de Cb i Cr 128, per tal de desfer la transformació feta en la compressió.

En sisè lloc, retornem els valors del format YCbCr a RGB, i comprovem que aquests no es surten del rang(0, 255), ja que al arrodonir, seria possible trobar algun valor negatiu o major de 255.

En setè lloc, ajuntem els valors dels píxels respectant el format .ppm, i els retornem com una cadena de bytes.

En aquest algorisme, treballem principalment amb matrius d'enters i decimals, que manipulen els diferents components de cada píxel, i en alguns casos amb vectors que representen la matriu de forma plana. També utilitzem els arbres binaris, i les cues de prioritats, en el cas de la compressió amb Huffman. També cal remarcar l'ús de cadenes de bytes i booleans, per gestionar la entrada sortida. Però no es fan servir altres estructures de dades més enllà d'aquestes bàsiques.