

# Documentació UML

## Casos d'ús

**Comprimir un arxiu:** *L'usuari pot escollir a través de l'explorador d'arxius l'arxiu desitjat per a comprimir. Després tindrà la possibilitat d'escollir un algorisme de compressió dels disponibles, algoritme LZ78, algoritme LZW, algoritme LZSS, algoritme JPEG, i en el cas que l'algorisme escollit sigui JPEG es podrà escollir el nivell de compressió. Seguidament podrà escollir el directori del disc on es guarda l'arxiu comprimit (per defecte es guarda al mateix directori de l'arxiu sense comprimir), a més al final de la compressió se li mostraran per pantalla les estadístiques locals de la compressió actual, que són temps de compressió, grau de compressió, velocitat de compressió, mida de l'arxiu inicial, mida de l'arxiu final.*

**Comprimir carpeta/conjunts de fitxers:** *L'usuari pot escollir a través de l'explorador d'arxius un conjunt d'arxius o una carpeta. Després l'algoritme de compressió se selecciona automàticament per a cada arxiu. Seguidament podrà escollir el directori del disc on es guarda l'arxiu comprimit (per defecte es guarda al mateix directori de l'arxiu sense comprimir), a més al final de la compressió se li mostraran per pantalla les estadístiques locals de la compressió actual, que són temps de compressió, grau de compressió, velocitat de compressió, mida de l'arxiu inicial, mida de l'arxiu final.*

**Descomprimir:** *L'usuari escull l'arxiu o carpeta a descomprimir a través de l'explorador d'arxius i es descomprimeix fent servir l'algoritme de compressió adequat, a més al final de la descompressió se li mostraran per pantalla les estadístiques locals de la descompressió actual, que són temps de descompressió, grau de descompressió, velocitat de descompressió, mida de l'arxiu inicial, mida de l'arxiu final.*

**Generació estadístiques globals:** *L'usuari podrà accedir a les estadístiques de totes les compressions fetes per l'aplicació a partir d'un bot. Les estadístiques globals a les quals tindrà accés seran les estadístiques locals en mitjana, el nombre total d'arxius, el nombre de compressions per cada algoritme específic.*

**Algoritme LZ78:** *Algoritme lossless per comprimir i descomprimir textos. Aquest algoritme està especialitzat per l'ús de textos, encara que pot comprimir imatges.*

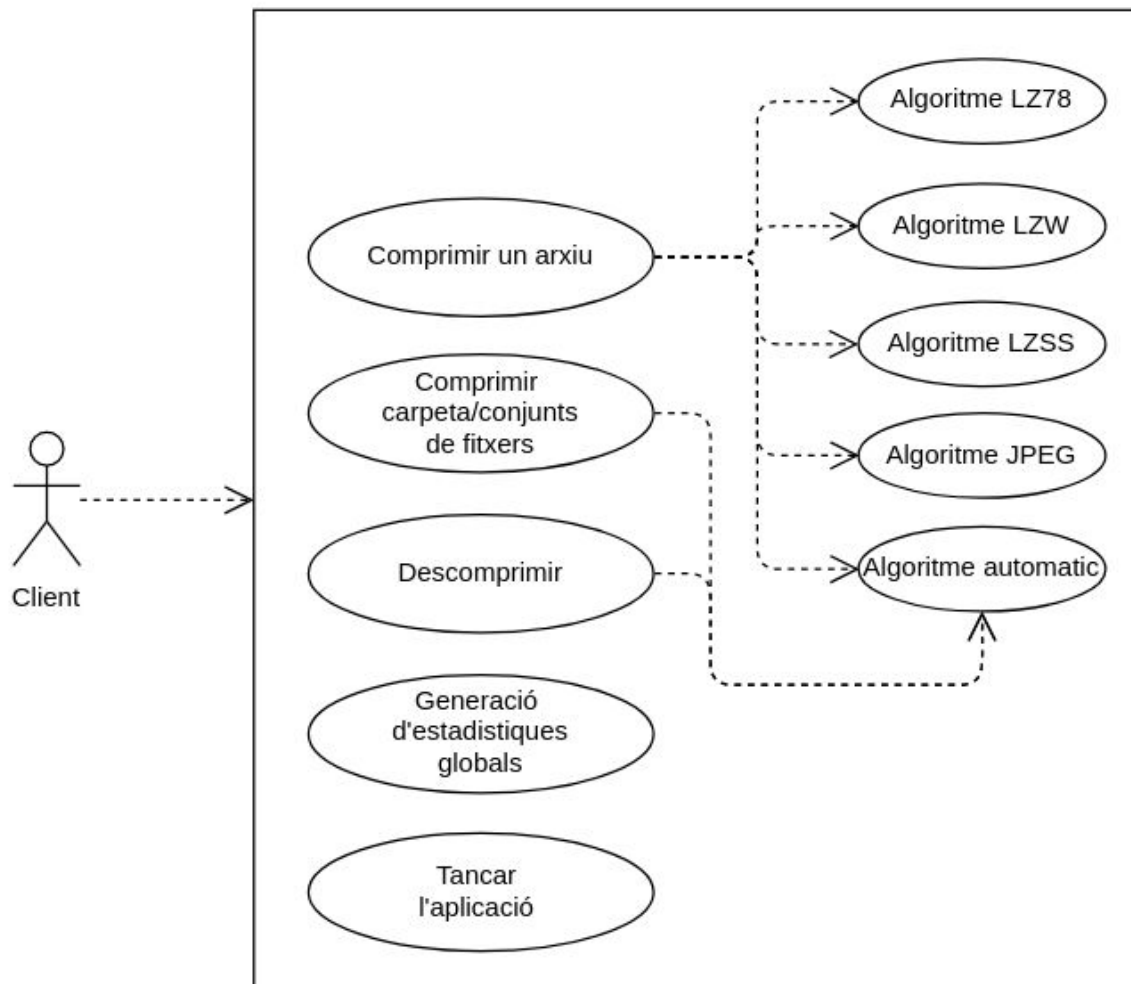
**Algoritme LZW:** *Algoritme lossless per comprimir i descomprimir textos. Aquest algoritme està especialitzat per l'ús de textos, encara que pot comprimir imatges.*

**Algoritme LZSS:** *Algoritme lossless per comprimir i descomprimir textos. Aquest algoritme està especialitzat per l'ús de textos, encara que pot comprimir imatges.*

**Algoritme JPEG:** *Algoritme lossy per comprimir i descomprimir imatges. Aquest algoritme està especialitzat per l'ús d'imatges. Solament accepta un rang de color de 0..255.*

**Automàtic:** El sistema automàticament escull l'algoritme més eficient per cada arxiu introduït i el tornarà comprimit en el cas que s'hagi de comprimir o descomprimit en el cas que s'hagi de descomprimir.

**Tancar aplicació:** L'usuari decideix tancar l'aplicació.



## Diagrama de classes\*

**Estadística:** *La classe estadística es la classe encarregada de gestionar les estadístiques sobre la compressió i descompressió d'arxius.*

**Estadística global:** *Les estadístiques globals són aquelles que representen una col·lecció de totes les compressions i descompressions que s'han fet al llarg de l'execució del programa.*

**Algoritme:** *La classe algoritme és la representant dels diferents algorismes de compressió i és l'encarregada de comprimir o descomprimir un arxiu específic.*

**Huffman:** *La classe Huffman es un altre tipus d'Algoritme de compressió pero amb una implementació diferent que està especialitzada per ser una extensió de l'algoritme JPEG.*

**Utils:** *La classe utils representa un conjunt de funcionalitats que serveixen en la compressió i descompressió dels diferents algorismes.*

**LZW:** *Representa un tipus d'algoritme que s'utilitza per comprimir arxius.*

**LZ78:** *Representa un tipus d'algoritme que s'utilitza per comprimir arxius.*

**LZSS:** *Representa un tipus d'algoritme que s'utilitza per comprimir arxius.*

**JPEG:** *Representa un tipus d'algoritme que s'utilitza per comprimir arxius.*

**\*Diagrama adjunt en un pdf**

## Relació classes implementades

	Adrián Álvarez Martínez	Jaume Bernaus Serra	Pol Monroig Company	David Santos Plana
LZ78	-	-	Sí	-
LZW	-	Sí	-	-
LZSS	-	-	-	Sí
JPEG	Sí	-	-	-
Utils	Sí	-	Sí	-
Stats	-	-	Sí	-
GlobalStats	-	-	Sí	-
Algorithm	-	-	-	Sí
Huffman	Sí	-	-	-
DataCtrl	-	-	Sí	-
DomainCtrl	-	-	-	Sí
PresentationCtrl	-	-	-	Sí
DriverLZ78	-	-	Sí	-
DriverLZW	-	Sí	-	-
DriverLZSS	-	-	-	Sí
DriverJPEG	Sí	-	-	-
DriverHuffman	Sí	-	-	-
DriverDataCtrl	-	Sí	-	-
DriverDomainCtrl	-	Sí	-	-
DriverUtils	-	-	Sí	Sí
DriverStats	Sí	-	-	-
DriverGlobalStats	Sí	-	-	-

## Descripció algorismes implementats

### **LZ78:**

La idea de tots els algorismes Lempel-Ziv és que si un text no és uniformement aleatori llavors una paraula que ja hem vist és més probable que aparegui que una que no hem vist mai. Específicament LZ78 funciona construint un diccionari de paraules que s'anomenen frases, una frase és una concatenació de caràcters de mida més gran o igual a un. L'algoritme comença agafant la frase més petita del text que no haguem vist abans. Com que inicialment el diccionari està buit, la primera frase serà el primer caràcter. Seguidament agafem la següent frase que no haguem vist, de tal manera que si ja hem vist una frase el que fem és afegir el següent caràcter, fins que aquesta no estigui present al diccionari. Per cada frase  $w$  que no haguem vist la podem subdividir en dues sub-frases tal que  $w=xy$  i  $|y|=1$ . Això significa que si  $|w|=1$ , una frase format per un sol caràcter, tenim que  $x$  serà la paraula buida, que per precondició és una frase que sempre hem vist. Seguidament per cada paraula  $w$  que guardem al diccionari guardarem el índex de la posició de la paraula  $x$ . Quan acabem, la compressió representarà una sèrie d'índexs i caràcters. Per descomprimir simplement haurem de reconstruir el diccionari i les frases de manera recursiva. S'ha utilitzat un diccionari perquè hem de poder buscar i guardar frases úniques.

### **LZW:**

Com la majoria dels mètodes de compressió l'algoritme LZW identifica les cadenes de caràcters repetides per tal de crear una taula d'equivalències i pode'ls-hi assignar codis més breus.

L'algoritme del mètode LZW en una sola passada pel text pot crear el diccionari i codificar el text. A més no és necessari passar aquest diccionari junt amb la codificació ja que el diccionari es reconstrueix durant el procés de descompressió.

Això passa degut a que el diccionari comença inicialitzat amb 256 entrades. Es van afegint codificacions per cada parella de caràcters consecutius i més endavant de  $n$  caràcters, tot hi que aquesta codificació només s'acabara utilitzant si és torna a trobar aquests grups de caràcters en el text i d'aquesta forma s'aconseguiria la compressió. He decidit codificar els codis en 16 bits per tal de poder disposar de 65536 entrades en el diccionari.

Les estructures de dades per implementar l'algoritme han sigut un Map pel diccionari, ja que per cada entrada s'ha de guardar la cadena de caràcters, el seu codi i s'ha de poder buscar si el codi ja existeix.

Un string per tal de guardar els codis en binari de l'output o el text descomprimit en el cas de la descompressió. I utilitza strings per a llegir els caràcters i operar amb ells creant les cadenes.

### **LZSS:**

Aquest algoritme consisteix a emmagatzemar els 4096 caràcters més recents del text a descomprimir, i a mesura que es llegeixen els caràcters buscarem coincidències de la seqüència de caràcters nous amb els ja llegits, si coincideixen com a mínim 3 caràcters nous, i com a màxim 18, amb els ja llegits codificarem aquesta cadena de caràcters com a dos bytes que indiquen la posició en la que es troba la coincidència amb el buffer on tenim emmagatzemats els ja llegits i indicaran també la longitud de caràcters que s'han repetit, i d'aquesta manera disminuir la mida del fitxer, per tal de poder descomprimir després el fitxer tindrem un byte cada certa mida, els bits del qual valdran 1 o 0, que cada bit indica si es llegirà un caràcter que es pot escriure directament, o si haurem de llegir dos bytes per a obtenir la seqüència codificada explicada anteriorment, i d'allí obtenir els caràcters que hi havien escrits originalment.

L'estructura de dades d'aquest algoritme està tota feta amb vectors de bytes, 3 vectors que representen un arbre binari, que intentarà ser complet, el qual indicarà les diferents seqüències de caràcters començades per un determinat caràcter dintre de 256 possibilitats, els possibles en ASCII, que s'hàn anat trobant a mesura que s'executa el codi. Hi ha un vector que consisteix en un buffer que emmagatzema els 4096 caràcters més recents, i un altre vector on es guardarà tota la codificació del codi que retorna la funció, en comprimir o en descomprimir.

### JPEG:

Aquest algorisme consisteix en un algorisme de compressió d'imatges lossy, és a dir, que part de la informació de la imatge es perd en el procés, però és tan insignificant la pèrdua que els nostres ulls no la detecten. Principalment l'algoritme està basat en matrius, ja que la imatge en si és una matriu, i serà molt més fàcil treballar amb aquesta estructura de dades. Llavors, el que primer fa l'algoritme és llegir el ppm (és la versió P6), el que hem llegit són els colors dels píxels en RGB. En una imatge, hi existeix molta correlació entre aquests components de color, i nosaltres volem reduir la grandària de la imatge que és el principal objectiu, per tant haurem de fer una color space transform a YCbCr, que representen lluminositat(Y), el color cromàtic blau (Cb) i el color cromàtic vermell (Cr). La lluminositat descriu la brillantor de la imatge, mentrestant els altres dos components tenen com informació la tonalitat de la imatge. Així que ara tindrem 3 components per treballar. Ara tractarem individualment les components per a després agrupar-les un altre cop. Seguidament, per poder continuar amb l'algoritme, haurem de dividir els colors en blocs de 8x8. A continuació, farem el que es coneix com The Discrete Cosine Transform què és el cor del JPEG, on la DCT es tracta d'expressar una seqüència finita de números com a resultat de la suma de diferents senyals sinusoidals.

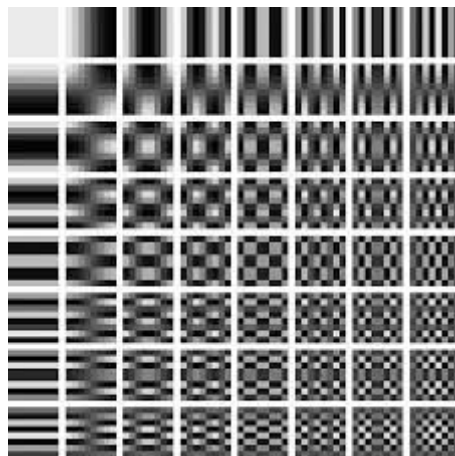


Figura 1: Seqüència de senyals de la DCT

Com podem observar en la Figura 1, tenim el bloc 8x8 amb les senyals pertinents de la DCT, i com la interpretem? Doncs si observem el punt 0.0, el que és totalment blanc, hi existeix un coeficient molt gran, vol dir que aquesta senyal apareix molt per tal de aconseguir aquesta senyal sinusoidal, si ens fixem, ens donarem compte que contra més complexa és una senyal el coeficient serà molt més petit. Una cosa molt important la DCT treballa en un rang de 1 a -1, ja que així aquesta forma la funció del sinus, llavors per tal de poder aplicar correctament la DCT haurem de normalitzar els components i ho farem restant la 128 (perquè és la meitat del color màxim de la imatge), i així aconseguirem normalitzar els components.

Com podem observar els coeficients són nombres reals que seran guardats com enters, això significa que haurem d'arrodonir els coeficients, així que primerament dividirem els coeficients pels factors de quantització i després arrodonirem:  $\text{round}(F_{w,u} / Q_{w,u})$ . Això ens permet poder ressaltar els senyals més presents per tal de formar aquest senyal sinusoidal. A més, aquesta part efectua la qualitat de la imatge JPEG.

Seguidament, per poder codificar correctament, haurem d'ordenar els coeficients degudament, ja que si anem cap a la dreta, incrementarà el senyal horitzontal, i si ens movem cap a baix el senyal vertical. Llavors ens haurem de moure en zig-zag per ordenar els coeficients.

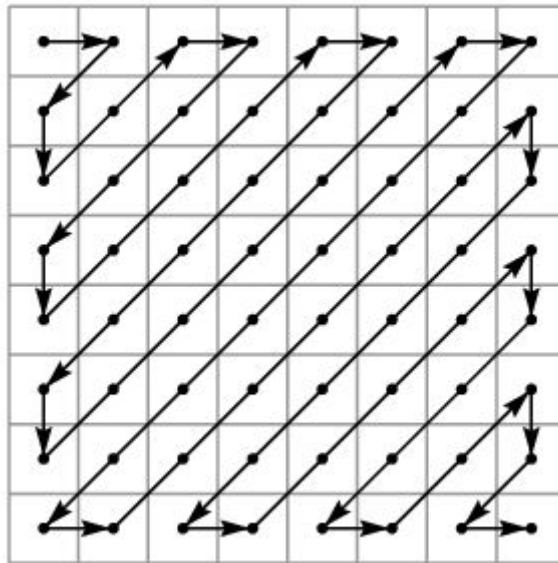


Figura 2: Recorregut en zig-zag

Així, ara, ja tenim preparat per poder codificar la imatge amb Huffman, on compte el nombre de freqüències dels coeficients, és a dir, quantes vegades apareixen, per tal d'ordenar-les en un arbre binari per aconseguir un codi binari, on contra la freqüència sigui més alta, més curt serà el codi i viceversa. Finalment, quan aconseguim els 3 components codificats ho escriurem en l'arxiu JPEG, i ja podrem donar per finalitzada la fase de compressió.

Per altra part per fer la descompressió, haurem de fer tot a l'inversa, és a dir haurem de desfer Huffman seguint l'arbre creat, després haurem de tornar a recórrer en zig-zag el bloc de 8x8, i desquantitzar amb el mateixos factors (multiplicar en comptes de dividir), i haurem d'aplicar la Inverse Discrete Cousin Transform on bàsicament desfarem el DCT abans fet, i normalitzarem a 0..255, sumant-li 128 als coeficients. Una vegada fet això faltaria l'últim pas, passar de YCbCr a RGB, i escriure en el fitxer .ppm.

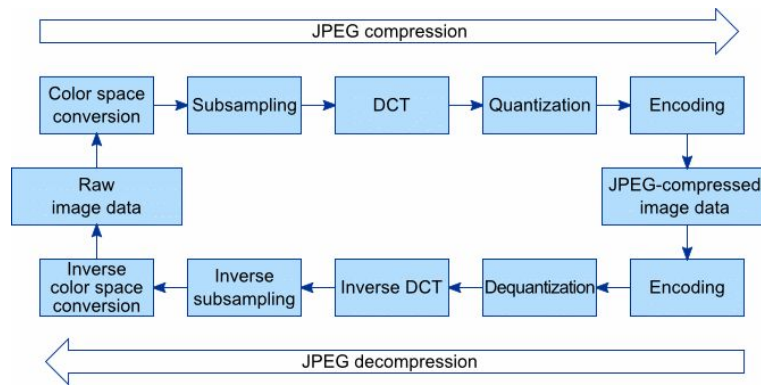


Figura 3: Esquema de la compressió i descompressió del JPEG

En aquesta primera entrega no s'ha arribat a assolir la visualització de l'arxiu comprimit(.jpeg) ni la visualització correcta de l'arxiu descomprimit (.ppm), però queda remarcar que aquestes funcionalitats s'intentaran aplicar de cara a la segona entrega.