PROJECTE DE PROGRAMACIÓ

1er lliurament

Mireia Bosque Marí mireia.bosque

Marina Díaz Reyes marina.diaz

Pablo Ortiz López pablo.ortiz.lopez

Xavier Valls Ruiz xavier.valls.ruiz

Nombre del grup: 5.5

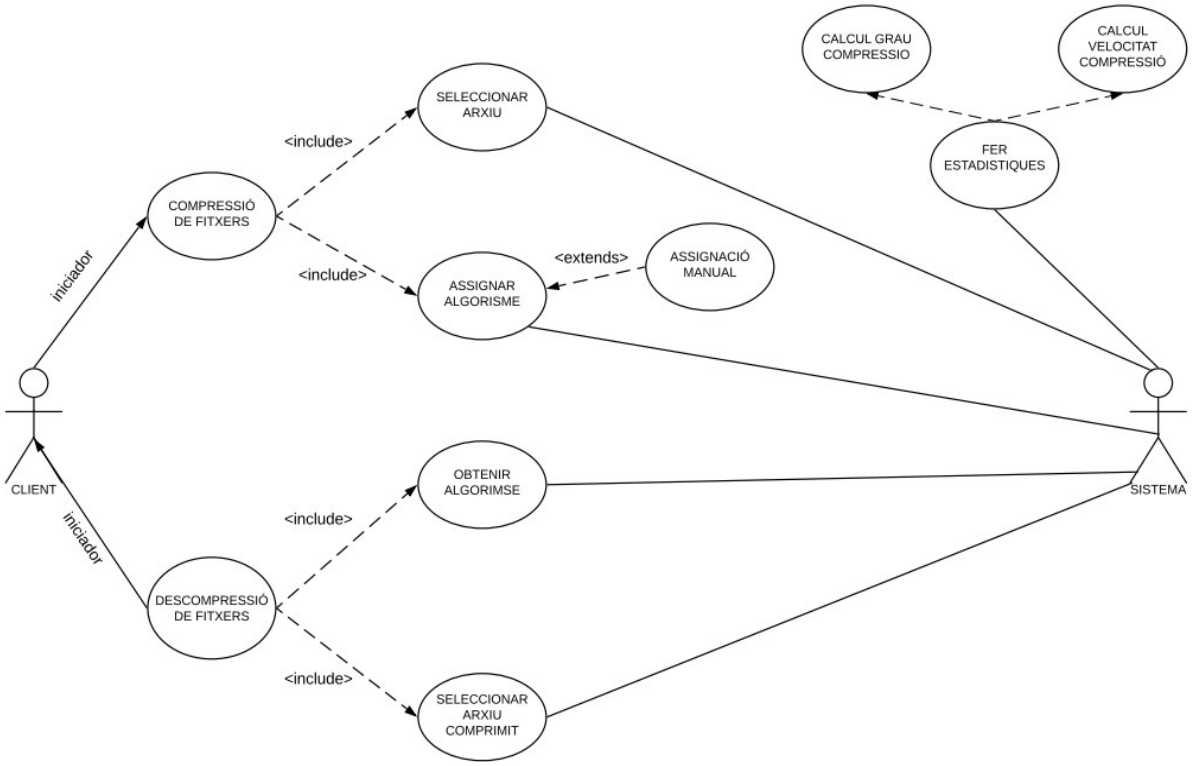
Versió: 1.0

Data de lliurament: 18/11/2019

ÍNDEX

1. Definició dels casos d’ús

1.1. Diagrama dels casos d’ús



1.2. Descripció detallada casos d’ús

Nom Cas: Comprimir fitxers

Actor: client

Escenaris d'èxit: Comprimir un fitxer/carpeta segons un cert algorisme

Escenaris erronis: - no tenir permisos d'escriptura a la carpeta destí

- no existeix fitxer/carpeta seleccionat

Nom Cas: Descomprimir fitxers

Actor: client

Escenaris d'èxit: Comprimir un fitxer/carpeta segons un l’algorisme que el va comprimir

Escenaris erronis: algorisme donat incorrecte

Nom Cas: Assignar algorisme
Actor: sistema
Escenaris d'èxit: escull l'algorisme més eficient per comprimir el tipus de fitxer donat
Escenaris erronis: -

Nom Cas: Assignar algorisme manual
Actor: client
Escenaris d'èxit: comprimir fitxer amb l'algorisme donat
Escenaris erronis: - que sigui un .txt i l'algorisme donat sigui Jpeg
- que sigui un .ppm i que l'algorisme donat sigui != Jpeg

Nom Cas: Fer estadístiques
Actor: sistema
Escenaris d'èxit: obtenir una mitjana del grau de compressió i velocitat per cada tipus d'algorisme.
Escenaris erronis: -

Nom Cas: Calcular grau de compressió
Actor: sistema
Escenaris d'èxit: calcular el grau de compressió de un fitxer segons un algorisme
Escenaris erronis:

Nom Cas: Seleccionar arxiu
Actor: client
Escenaris d'èxit: determinar els fitxers/carpeta que el client te intenció de comprimir
Escenaris erronis: -no existència de l'arxiu seleccionat
- l'arxiu/carpeta ja està comprimit
- Intentar seleccionar un tipus de fitxer erroni

Nom Cas: Calcul velocitat de compressió

Actor: sistema

Escenaris d'èxit: Retorna la velocitat de comprimir un arxiu.

Escenaris erronis:

Nom Cas: Obtenir algorisme

Actor: sistema

Escenaris d'èxit: retorna l'algorisme en que es va comprimir l'arxiu

Escenaris erronis: - no tenir disponibilitat d'aquesta informació

- rebre informació invalida.

Nom Cas: Seleccionar arxiu comprimit

Actor: client

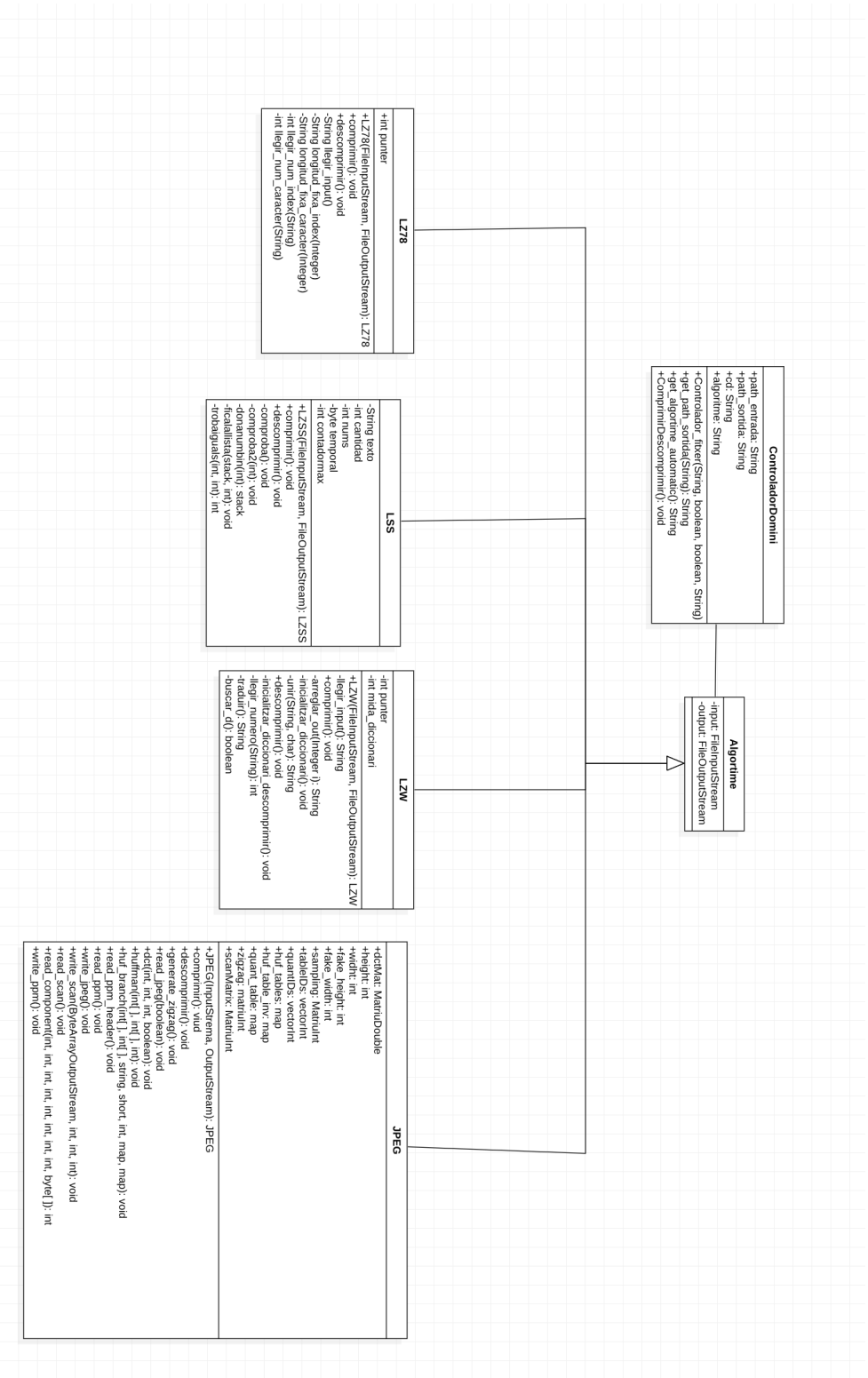
Escenaris d'èxit: Determinar arxiu que el client vol descomprimir.

Escenaris erronis: - que l'arxiu no hagi estat comprimit

- tipus erroni
- no existeix arxiu

2. Model conceptual de les dades

2.1. Diagrama UML estàtic complet



2.2. Especificació detallada

2.2.1. Implementació LZ78

2.2.1.1. Atributs de la classe

- private FileInputStream input = null; //Fitxer d'entrada, aquí tindrem la cadena de text que voldrem comprimir/descomprimir.
- private FileOutputStream output = null; //Fitxer de sortida, aquí tindrem la cadena de text que voldrem comprimir/descomprimir.
- int punter; //Enter on es guarda la posició del String que la que s'està llegint.

2.2.1.2. Creadora de classe

- LZ78(InputStream input, FileOutputStream output)
  - Pre: rep un FileInputStream del qual llegirà la cadena a processar i un FileOutputStream en el qual escriurà les dades un cop processades.
  - Post: els atributs privats input i output fan referència als objectes que entren per paràmetre.

2.2.1.3. Mètodes públics

- public void comprimir()
  - Pre: l'objecte LZ78 disposa d'un input i output no nuls.
  - Post: si input és un arxiu .txt vàlid, s'escriu un fitxer .lz78 equivalent i vàlid al FileOutputStream referenciat per output.

- public void descomprimir()
  - Pre: l'objecte LZ78 disposa d'un input i output no nuls.
  - Post: si input és un arxiu .lz78 vàlid, s'escriu un fitxer .txt equivalent i vàlid al FileOutputStream referenciat per output.

2.2.1.4. Mètodes privats

2.2.1.4.1. Funcions comuns

- private String llegir_input()
  - Pre: -
  - Post: s'obté un String amb el text que conté l'input codificat en Unicode de 16 bits.

2.2.1.4.2. Funcions per a la compressió

- private String longitud_fixa_index(Integer i)
  - Pre: i no és nul.
  - Post: s'obté l'enter entrat per paràmetre codificat amb longitud fixa de 6 dígits.

- private String longitud_fixa_caracter(Integer i)
  - Pre: i no és nul.
  - Post: s'obté l'enter entrat per paràmetre codificat amb longitud fixa de 3 dígits.

2.2.1.4.3. Funcions per a la descompressió

- private int llegir_num_index(String text_in)
  - Pre: text_in és no nul.
  - Post: s'obté el número equivalent a la cadena de 6 caràcters entrada per paràmetre.

- private int llegir_num_caracter(String text_in)
  - Pre: text_in és no nul.
  - Post: s'obté el número equivalent a la cadena de 3 caràcters entrada per paràmetre.

ANOTACIONS: El mètode comprimir() treu com a sortida una cadena de nombres formada per índex i caràcters, on el primer es codifica amb 6 dígits i el segon amb 3. El caràcter com màxim pot estar codificat amb 3 dígits perquè la codificació en ASCII arriba com a màxim al valor 255, amb la qual cosa no hi ha problema possible. En el cas de l'enter, en canvi, podia tenir el problema de quedar-me sense espai (ja que el diccionari que utilitzo em permet indexar fins a 2^{32} elements perquè treballo amb enters), amb la qual cosa havia de decidir quin nombre de dígits utilitzar.

Com que amb entrades petites l'algoritme no comprimeix (tema comentat al fitxer Jocs_de_prova_LZ78.txt), he preferit utilitzar 6 dígits en comptes de 5. (Amb 5 dígits i entrades més grans comprimia bé però vaig comprovar que no li faltava gaire per a quedar-se sense espai). Fent-ho d'aquesta manera m'asseguro de poder entrar fitxers grans, que és la manera de comprovar que l'algoritme realment comprimeix: entrant-li un text de 2,3MB per exemple obtinc un comprimit de 1,7MB.

2.2.2. Implementació LZSS

2.2.2.1. Atributs de la classe

- private FileInputStream input = null; Fitxer de entrada que serà lo que vols comprimir/descomprimir.
- private FileOutputStream output = null; Fitxer de sortida on anirà el fitxer de entrada comprimit/descomprimit.
- private String texto = ""; És al String on quan comprimeixo fico el text del input. I a descomprimir és on fico eltext descomprimit
- private int cantidad = 1; Variable que apunta al bit del byte que toca tractar.
- private int nums = 0; Variable que indica quants números té en binari el número que es vol afegir al output de descomprimir
- private byte temporal = 0; Byte on omplim el que s'ha de posar el output per poder fer-lo bit a bit a comprimir

- private int contadormax = 0; És una variable utilitzada a comprimir per indicar quin és el màxim número de coincidències amb parts anteriors per cada lletra.

2.2.2.2. Creadora de classe

- public LZSS(FileInputStream input, FileOutputStream output)
  - **Pre:** Rep un FileInputStream d'on llegirà la cadena a processar i un FileOutputStram en el qual escriurà les dades un cop processades.
  - **Post:** Els atributs inputs i outputs fan referència als que entren com paràmetre.

2.2.2.3. Mètodes públics

- public void comprimir()
  - **Pre:** L'objecte LZSS disposa d'un input i output no nuls. I input és un arxiu txt.
  - **Post:** Escriu al output un arxiu .lzss equivalent i vàlid.
- public void descomprimir()
  - **Pre:** L'objecte LZSS disposa d'un input i output no nuls. I input és un arxiu lzss.
  - **Post:** Escriu un arxiu .txt al output equivalent i vàlid.

2.2.2.4. Mètodes privats

2.2.2.4.1. Funcions de compressió

- private void comrpoba()
  - **Pre:**
  - **Post:** Suma 1 a cantidad, si aquesta arriba a 9 augmenta puntero i cantidad el col·loca a 1, a més escriu al output el byte temporal.
- private Stack donanumbin(int numdec)
  - **Pre:** numdec no és null i és un número decimal.
  - **Post:** Retorna una pila amb numdec passat a binari ordenat amb el que té més pes a dalt de tot, també modifica nums i li assigna el tamany de la pila.

- private voi ficalallista(Stack numsbin , int tipo)
  - **Pre:** numsbin no està buida i és un número binari ordenat de més a menys pes, tipo no és null. El tamany de numsbin és menor o igual al valor de tipo.
  - **Post:** S'afegeixen els bits de numsbin a temporal, si el tamany de numsbin és menor que tipo la diferència s'omple de 0.
- private voi trobaiguals(int i, int j)private void comrpoba()
  - **Pre: -**
  - **Post:** Suma 1 a cantidad, si aquesta arriba a 9 augmenta puntero i cantidad el col·loca a 1, a més escriu al output el byte temporal.
- private Stack donanumbin(int numdec)
  - **Pre:** numdec no és null i és un número decimal.

- **Post:** Retorna una pila amb numdec passat a binari ordenat amb el que té més pes a dalt de tot, també modifica nums i li assigna el tamany de la pila.

- private voi ficalallista(Stack numsbin , int tipo)
  - **Pre:** numsbin no està buida i és un número binari ordenat de més a menys pes, tipo no és null. El tamany de numsbin és menor o igual al valor de tipo.
  - **Post:** S'afegeixen els bits de numsbin a temporal, si el tamany de numsbin és menor que tipo la diferència s'omple de 0.

- private voi trobaiguals(int i, int j)
  - **Pre:** i i j no són nulls ni iguals.
  - **Post:** Busca cadenes de caràcters iguals, va mirant des de i, i de j i va mirant quants seguits coincideixen a ambdos costats i apunta el valor de quants igual hi ha. Si aquest contador és més gran que el contadormax actualiztem el contadormax amb aquest valor i retornem la r que serà la j inicial, si no es major la r serà -1.

2.2.2.4.2. Funcions de descompressió

- private void comproba2(int puntero)
  - **Pre:** puntero no null.
  - **Post:** Suma 1 a cantidad, si aquesta arriba a 9 augmenta puntero i cantidad el col·loca a 1.

2.2.3. Implementació LZW

2.2.3.1. Atributs de la classe

- **FileInputStream:** Byte Stream del arxiu d'entrada
- **FileOutputStream output :** Byte Stream del arxiu de sortida
- **Punter:** iterador pe recórrer el text d'entrada, identifica fins on s'ha llegit
- **Mida_diccionari:** indica la mida del diccionari en cada moment.

2.2.3.2. Creadora de classe

- Jzw(InputStream input, FileOutputStream output)
  - **Pre:** reb un InputStream del qual llegirà, i un FileOutputStream al qual escriurà les dades
  - **Post:** les variables privades input y output fan referencia als objectes anteriorment esmentats.

2.2.3.3. Mètodes públics

- void comprimir()
  - **Pre:** L'objecte Jzw disposa d'un input y output no nuls
  - **Post:** Si input es un arxiu .txt vàlid, s'escriu un fitxer .lzw equivalent i vàlid, a l'output stream referenciat per output.
- void descomprimir()
  - **Pre:** L'objecte Jzw disposa d'un input i output no nuls
  - **Post:** Si input es un arxiu .lzw vàlid s'escriu a output un arxiu .txt amb el text equivalent.

2.2.3.4. Mètodes privats

2.2.3.4.1. Funcions comuns

- String Llegir_input():
  - **Pre:** -
  - **Post:** retorna un string amb el text que conté el input.

2.2.3.4.2. Funcions per la compressió

- void Inicialitzar_diccionari(map<String,Integer> diccionari):
  - **Pre:** diccionari inicialment buit.
  - **Post:** diccionari inicialitzat amb totes les cadenes de un sol caràcter.

- String Arreglar_out(Integer i):
  - **Pre:** i no null
  - **Post:** retorna un string amb el i augmentat a 5 digits.

- String unir(String s, char c):
  - **Pre:** c no null
  - **Post:** retorna un string amb la concatenació de s+c

2.2.3.4.3. Funcions per la descompressió

- void inicialitzar_diccionari_descomprimir(map<Integer,String> diccionari):
  - **Pre:** diccionari inicialment buit.
  - **Post:** diccionari inicialitzat amb totes les cadenes de un sol caràcter.

- int llegir_numero(String text_in):
  - **Pre:** text_in no null

- **Post:** s'obté la cadena dels 5 caràcters següents de la variable global punter del text_in.

- String traducir(Integer cod_viejo, map<Integer,String> diccionari):
  - **Pre: cod_viejo >= 0, diccionari no null**
  - **Post:** retorna el el valor de la clau cod_viejo del diccionari.

- boolean buscar_d(Integer cod_nuevo, map<String,Integer> diccionari):
  - **Pre:** cod_nuevo >= 0, diccionari no null
  - **Post:** retorna si cod_nuevo està al diccionari

2.2.4. Implementació JPEG

2.2.4.1. Atributs de classe

- InputStream input: Byte Stream del arxiu d'entrada
- FileOutputStream output : Byte Stream del arxiu de sortida
- double[][] dctMat: Matriu per realitzar la transformació DCT i DCT inversa.
- int height: Altura de la foto
- int width: Amplada de la foto
- int fake_height: Altura del scan JPEG
- int fake_width: Amplada del scan JPEG
- int[][] sampling: Configuració de mostreig de cada component de color
- int[] tableIDs: Taules de huffman associades a cada component
- int[] quantIDs: Taules de quantització associades a cada component
- Map<Integer, Map<String, Integer>> huf_tables: Diccionaris extrets dels arbres huffman
- Map<Integer, Map<Integer, String>> huf_tables_inv: Diccionaris extrets dels arbres huffman, invertits per la codificació

- Map<Integer, Integer[][]> quant_tables: Taules de quantització identificades pel seu ID.
- int[][] zigzag: Matriu de 63 posicions en 2 dimensions per recorrer una matriu 8x8 en zigzag-
- int[][][][][] scanMatrix: Matrius 8x8 de cadascun dels components de color.
- int[] offset: Darrer element DC escrit de cada component de color, per escriure la diferencia al pròxim MCU.

2.2.4.2. Creadora de classe

- Jpeg(InputStream input, FileOutputStream output)
  • Pre: reb un InputStream del qual llegirà, i un FileOutputStream al qual escriura les dades
  • Post: les variables privades input y output fan referencia als objectes anteriorment esmentats, es genera la matriu zigzag.

2.2.4.3. Mètodes públics

- void comprimir()
  • Pre: L'objecte Jpeg disposa d'un input y output no nuls
  • Post: Si input es un arxiu ppm vàlid, s'escriu un fitxer JPEG equivalent i vàlid, seguint l'estàndard JFIF, a l'output stream referenciat per output.

- void descomprimir()
  • Pre: L'objecte Jpeg disposa d'un input i output no nuls
  • Post: Si input es un arxiu JPEG vàlid que compleixi les següents característiques:
    ◦ Dues taules de quantització, la primera per Y y la segona per Cb i Cr
    ◦ Quatre taules Huffman, les dues primeres per Y y les dues segones per Cb i Cr
    ◦ Sampling de 2x2 per Y i 1x1 per Cb i Cr
    S'escriu a output un arxiu PPM amb la imatge equivalent.

2.2.4.4. Mètodes privats

2.2.4.4.1. Funcions comuns

- void generate_zigzag()
  - Pre: L'Array zigzag esta inicialitzada com una llista de posicions en 2 dimensions de longitud 63.
  - Post: L'Array zigzag conté, de forma ordenada, les posicions en 2 dimensions per recorrer una matriu 8x8 en forma de zigzag.

- void read_jpeg(boolean header)
  - Pre: Si header és fals, input ha de ser no nul.
  - Post: S'emmagatzemen a les variables privades de l'objecte els diversos atributs jpeg (taules, mida) del arxiu referenciat per input, o si header és cert, els continguts al arxiu "header.jpg". Finalitza un cop es troba el marcador de "scan".

- void dct(int component, int Y, int X, boolean inverse)
  - Pre: Es disposa d'una matriu 8x8 del component referenciat, a la posició x,y.
  - Post: S'aplica la transformació DCT a la matriu 8x8 referenciada, o DCT inversa si inverse és cert.

- void huffman(Integer[] counts, Integer[] values, int id)
  - Pre: counts i values no son nuls.
  - Post: Es guarda al diccionari de taules huffman, amb id 'id', la taula d'equivalencies resultat d'utilitzar 'counts' i 'values', de la mateixa manera que es fa a l'estàndard JPEG. També es guarda la taula d'equivalències inversa.

- int huf_branch(Integer[] counts, Integer[] values, String code, short pointer, int count, Map<String,Integer> table, Map<Integer,String> inv_table)
  - Pre: code no pot ser una String buida, tots els objectes d'entrada es troben definits.

• Post: Ambdues taules d'entrada disposen de totes les noves entrades trobades en aquesta branca de l'arbre, es retorna el nombre total d'elements ja inserits a la taula

2.2.4.4.2. Funcions per la compressió

- void read_ppm_header()
  • Pre: input no pot ser nul
  • Post: Si input fa referència a un arxiu PPM valid de tipus P6, es guarden les mides de la imatge a les variables privades, i es llegeix fins al punt on comencen les dades dels colors dels pixels.

- void read_ppm()
  • Pre: input fa referència a un arxiu PPM de tipus P6 vàlid, el seu punter de lectura es troba al inici de les dades de color dels pixels.
  • Post: S'escriuen les matrius 8x8 amb els samplings corresponents, de cada component per a la seva escriptura posterior al "scan" jpeg.

- void write_jpeg()
  • Pre: Les matrius 8x8 de scan estan preparades les seves dades corresponents
  • Post: S'aplica la transformació DCT a les matrius 8x8, es quantitzen, i s'escriu a un buffer temporal els bits de l'apartat scan. Després s'escriuen els bytes del header JPEG, injectant la nova mida de la imatge al apartat "Start of Frame".
  Finalment s'escriuen els bytes del "scan", obtinguts del buffer temporal anterior, insertant 0x00 després de cada 0xFF i afegint els zeros necessaris a l'últim byte.

- void write_scan(ByteArrayOutputStream scanBuffer,int component, int Y, int X)
  • Pre: scanBuffer no és nul, existeix la matriu 8x8 del component a la posició X,Y.
  • Post: S'escriu al buffer els bits necessaris per codificar aquesta matriu 8x8, tant el seu component DC com els seus components AC, emprant el paràmetre ZRL de la mateixa manera que a l'estàndard JFIF

2.2.4.4.3. Funcions per la descompressió

- void read_scan()
  • Pre: Input fa referència a un fitxer JPEG vàlid, i el seu punter de lectura es troba a l 'inici del bitstream del “scan”.
  • Post: S’escriuen les matrius 8x8 corresponents a cada component i posició a les variables privades.

- int read_component(int component, int tableID, int quantID, int x, int y, int scanH, int scanV, int pointer, byte[] scan)
  • Pre: pointer no és més gran que la mida de scan
  • Post: Es llegeix de scan, començant per pointer, la matriu 8x8 del component a la posició indicada, se li fa la DCT inversa i de-quantització. Retorna la proxima posició sense llegir del “scan”.

- void write_ppm()
  • Pre: Les matrius 8x8 necessàries estan definides
  • Post: Escriu un fitxer PPM de tipus P6 amb les dades disponibles a les matrius 8x8, fent la transformació a RGB a cada pixel.

3. Breu descripció dels algoritmes

3.1. Algoritme LZ78

L'algoritme LZ78 és un algoritme de compressió basat en la repetició de cadenes de caràcters, com més cadenes repetides hi hagi i com més llargues siguin aquestes, més es comprimirà l'arxiu.

3.1.1. Comprimir

Es llegeix el text d'entrada i es guarda codificat en Unicode de 16 bits. A continuació es llegeix aquest nou text obtingut caràcter per caràcter, i si el caràcter llegit no es troba indexat al diccionari (que inicialment està buit) s'afegeix i s'escriu al text de sortida juntament amb l'índex 0. En cas contrari, es guarda el caràcter en una cadena buida auxiliar i es llegeix el següent. Cada cop que es troba un caràcter ja guardat al diccionari, s'afegeix a la cadena. Quan es torna a trobar un caràcter que no és al diccionari, s'afegeix al diccionari la cadena formada per la cadena auxiliar i el nou caràcter trobat. A més a més s'escriu en el text de sortida l'índex del diccionari que referencia la cadena auxiliar, juntament amb el nou caràcter. En cas de que s'hagi acabat de llegir el text i ens quedin caràcters guardats a la cadena auxiliar, escriurem l'índex que referencia aquesta cadena.

3.1.2. Descomprimir

Igual que en la compressió, es llegeix el text d'entrada i es guarda codificat en Unicode de 16 bits. Tot seguit es llegeix el nou text obtenint la parella d'índex i caràcter. Si l'índex és 0 significa que el caràcter no es troba al diccionari i per tant cal afegir-lo, i s'escriu el caràcter en el text de sortida. En cas contrari, s'afegeix al diccionari la cadena a la que referencia l'índex unida amb el caràcter i s'escriu en el text de sortida aquesta nova cadena formada. En cas de que l'últim element llegit de la cadena sigui un índex i no un caràcter, simplement escriurem la cadena a la que referencia aquest.

3.2. Algoritme LZSS

Al algoritme que he programat li he donat les següents característiques, un flag 0 o 1 per saber si darrera va una lletra o dos números. Si el flag és 0, significa que els següents 8 bits seran un char, si el flag és 1 darrera vindran 13 bits per desplaçament i 5 per tamany.
Amb els 13 bits de desplaçament es podrà mirar fins un desplaçament de 8196 caràcters enrere i amb els 5 bits serem capaços d'agrupar en com a màxim 31 caràcters. Però com si agrupem menys de 3 caràcters estarem utilitzant 19 bits i fent-ho individualment utilitzaríem 9 o 18 bits com a mínim agruparem 3 lletres, per tant evitarem el agrupar 1 i 2 caràcters. Així doncs els 5 bits en serviran per codificar de tamany 3 a tamany 34.

3.2.1. Comprimir

Llegeix tots els caràcters del text, per cada caràcter menys el primer busca quants caràcters seguits es repeteixen a un altre part anterior del fitxer. Com el desplaçament són 13 bits si el caràcter que està mirant és més petit que 8196 busca coincidències fins al caràcter inicial, si és més gran busca fins el caràcter en el que està menys 8196.
Es queda amb la posició del caràcter on té el màxim número de coincidències i amb aquest número, sempre i quan no sigui major a 34.
Si el número màxim de coincidències és major que 2 fica el flag a 1 i afegeix el desplaçament i el tamany d'aquesta coincidència.
Per altra banda si el número màxim de coincidències és menor que 3 fica el flag a 0 i els 8 bits del caràcter.

3.2.2. Descomprimir

Va llegint el flag, si val 0 sabrem que darrera venen 8 bits que seran el caràcter per tant el següent flag serà 8 bits darrera, i si el flag és 1 vindran 13 de desplaçament i 5 de tamany i el següent flag estarà 18 bits després.
Per tant, quan val 0 llegeix els següents 8 bits els transforma a char i l'afegeix al String resposta.

Si val 1, llegeix la distància de 13 bits i el tamany de 5. I escriu al String resposta el match a aquesta distància i de tamany que acaba de llegir.

3.3. Algoritme LZW

L'algorisme LZW és un algorisme basat en la repetició de cadenes de lletres, com més cadenes repetides i com més llargues, més es comprimirà l'arxiu.

3.3.1. Comprimir

Primer de tot s'inicialitza el diccionari amb totes les cadenes d'un sol caràcter, després, es llegeix el text d'entrada i es guarda codificat en Unicode de 16 bits. A continuació es llegeix aquest nou text obtingut caràcter a caràcter. Si el caràcter llegit concatenat amb la cadena anterior (en cas de ser el primer amb una cadena buida) està al diccionari, simplement actualitzem la cadena anterior amb la concatenació, en cas contrari, traiem per la sortida el index corresponen al caràcter antic, actualitzem el diccionari amb aquesta nova cadena. Seguint aquests passos fins haver llegit el text complet, obtindrem el text comprimit.

3.3.2. Descomprimir

Igual que al comprimir, els dos primers passos són, inicialitzar el diccionari i llegir el text guardant-lo en un string. Llegeixo el primer index, en llegir-lo me'l guardo com a caràcter antic I com sé segur que estarà al diccionari, agafo el seu caràcter corresponent i el trec per la sortida i me'l guardo (caràcter = primer index), seguidament llegeixo tots els index fins acabar el text, un a un. Per cada element llegit comprovo si està al diccionari, en cas d'estar-ho em guardo a cadena el caràcter corresponent (cadena = caràcter nou), en cas contrari, em guardo a cadena el caràcter concatenat amb el caràcter corresponent al index anterior(cadena = caràcter + traducció index_antic). Finalment, trec per la sortida la cadena i afegeixo al diccionari el primer element de la cadena unit amb el caràcter corresponent al index antic.

3.4. Algoritme JPEG

3.4.1. Comprimir

Es llegeixen les taules de quantització i Huffman del header per defecte, que s'utilitzaran posteriorment.
Es llegeix l'arxiu ppm, i es van transformant les components rgb de cada pixel a YCbCr, i s'escriuen a la posició apropiada de scanMatrix.
Es fa la transformació dct a totes les matrius 8x8, i s'escriuen seguint l'algorisme establert al estàndard JFIF, quantitzant cada component, un cop acoblats tots els bits a un buffer de bytes, s'escriu el header, al qual només es canvien els 4 bytes de mida d'imatge, i després s'escriuen els bytes resultat de combinar tots els bits obtinguts prèviament, afegint 0x00 després de cada byte 0xFF, i afegint zeros als últims bits del darrer byte

3.4.2. Descomprimir

Es llegeixen les taules de Huffman i quantització, i posteriorment es llegeix l'scan, emprant les taules de huffman obtingudes. Un cop s'omplen les matrius de scanMatrix, s'aplica la dct inversa a cadascuna de les matrius i es dequantitzen.
Finament s'escriu l'arxiu ppm, començant per el header i posteriorment obtenint per cada pixel les components Y, Cb i Cr necessàries de scanMatrix, es transformen a RGB i s'escriuen a l'arxiu.

4. Llista de les classes implementades per cada membre del grup

- Les classes Main, Controlador_fitxer, Driver_LZ78, Driver_LZSS I Driver_LZW han estat implementades per Mireia Bosque, Marina Díaz i Xavier Valls.
- A més Marina Díaz ha fet la classe LZ78.
- Xavier Valls ha creat la classe LZSS.
- Mireia Bosque la classe LZW.
- I Pablo Ortiz ha creat les classes JPEG i JpegDriver

5. Relació de les llibreries externes utilitzades

- Java.util.Scanner: Utilitzada per poder llegir des de la consola.
- Java.io.IOException: Utilitzada per
- Java.io.FileInputStream: Utilitzada per llegir a través de FileInputStream a tots els algoritmes.
- Java.io.FileOutputStream: Utilitzada per escriure a través de FileOutputStream a tots els algoritmes.
- Java.io.BufferedReader: Utilitzada als drivers per llegir els fitxers.
- Java.io.FileReader: Utilitzat per comprovar que a diferents fitxer hi ha el mateix als divers.
- Java.io.InputStreamReader: Serveix per llegir i l'utilitzem per fer que funcioni amb fitxers amb caràcters especials.
- Java.util.Stack: Utilitzada al algoritme LZSS per poder utilitzar piles.
- Java.util.HashMap: Utilitzada a LZW per utilitzar hashmap.
- Java.util.Map: Utilitzada a LZW per utilitzar diccionaris.
- Java.io.ByteArrayOutputStream: Utilitzada a JPEG per crear un array de bytes.
- Java.util.TreeMap: Utilitzat a JPEG.