# PROTECCIÓ DEL COPYRIGHT

**ESTUDIANT:**
**JORDI TRUBAT DALMAU (ETIS)**

**CONSULTOR:**
**ANTONI MARTÍNEZ BALLESTÉ**

**10 DE JUNY DE 2004**

*Imatge marcada de Lenna*

## ÍNDEX DE CONTINGUTS

1. Introducció ..... 2
2. Estat de l'art ..... 3
3. Objectius del TFC ..... 3
4. Enfocament i mètode seguit ..... 4
5. Planificació del projecte ..... 5
6. Disseny del sistema ..... 7
7. Implementació del sistema ..... 13
8. Instal·lació del programari ..... 17
9. Manual d'usuari ..... 17
10. Joc de proves ..... 22
11. Conclusions ..... 25
12. Glossari ..... 26
13. Referències .....27
14. Annex. Llistats dels fitxers font..... 29

## ÍNDEX DE FIGURES

Algorisme d'encast de la marca ..... 8
Algorisme de confabulació .....11

## 1. INTRODUCCIÓ

La irrupció de les noves tecnologies ha convertit en una trivialitat la duplicació i disseminació de textos, imatges, sons i vídeos digitals. Quan aquests mitjans estan protegits per un copyright es presenta el problema de la seva protecció. Una tècnica per marcar el mitjà digital i així poder identificar el seu creador consisteix en enterrar una marca digital en el mateix que resulti imperceptible pel comprador, però que identifiqui de manera unívoca l'autor del producte. Aquesta tècnica s'anomena *watermarking*. Observem que el *watermarking* permet, en principi, identificar l'autor, però no el comprador ni molt menys impedir les còpies il·legals del producte. Una segona tècnica relacionada amb l'anterior és el *fingerprinting*. A diferència del *watermarking*, aquesta darrera tècnica consisteix en enterrar una marca en el producte digital que identifica el comprador del producte. Si el comprador decideix fer-ne còpies il·legals, totes les còpies duran la mateixa marca i, en principi, hom pot identificar l'impostor.

Per altra banda, el comprador pot intentar esborrar o emmascarar la marca, ja sigui sotmetent el producte digital a alteracions o confabulant amb altres compradors. Resulta, doncs, important que la tècnica utilitzada per marcar el producte digital sigui prou robusta per extreure la marca en casos en que aquest ha sofert modificacions i/o confabulacions.

L'objectiu d'aquest TFC, doncs, és implementar un sistema senzill de protecció del copyright pel cas d'imatges de format BMP mitjançant un esquema de *fingerprinting* basat en codis duals de Hamming que permeten recuperar la marca quan la imatge marcada ha estat sotmesa a certs tipus de processament digital i en el cas de confabulació de dos compradors.

## 2. ESTAT DE L'ART

Cronològicament, les primeres aproximacions per enterrar la marca que identifica el comprador o el venedor en un mitjà digital (imatge, so) han tingut lloc en el domini espacial del mitjà. En concret, s'utilitza els bits de menys pes per codificar la marca. Una alternativa millor, en el cas de les imatges, consisteix en codificar la marca en les diferències de valor entre un cert nombre de parells de píxels. Un esquema més robust per a imatges, vídeo i so es basa en el mètode espectral que consisteix en enterrar la marca en aquelles regions del conjunt de dades que presenten un tret sobresortint en el diagrama espectral del conjunt de dades. És a dir, si en el primer cas la tècnica de marcatge té lloc en el domini espacial, aquí el marcatge té lloc en el domini freqüencial. Algunes tècniques més avançades fan ús de les dues aproximacions. Per exemple, en el cas de les imatges, hom pot escollir aquelles regions del domini espacial de la imatge que mostren una riquesa de detall més gran i després enterrar la marca en aquestes regions utilitzant el domini freqüencial. Recentment, alguns investigadors han començat a estudiar també la manera de marcar figures tridimensionals.

## 3. OBJECTIUS DEL TFC

En aquest TFC estudiarem un senzill sistema de marcatge en l'àmbit espacial per a imatges de tipus BMP amb codis duals de Hamming que dona bons resultats quan la imatge marcada ha estat sotmesa a un cert nombre limitat de processaments digitals i que ofereix robustesa total en el cas de confabulació de dos compradors. La idea consisteix en modificar lleugerament la imatge original tot inserint en la mateixa un patró de bits (marca) que pugui identificar de manera unívoca l'autor de la imatge o bé el seu comprador. La marca de la imatge original hauria de complir els requisits següents:

- Hauria de ser imperceptible. És a dir, un observador no ha de notar que la imatge ha estat modificada. Idealment l'observador no hauria de notar cap diferència entre la imatge original i la marcada.

- Hauria de ser impossible d'eliminar-la sense degradar la imatge fins l'extrem de que aquesta esdevingués inservible.

- Hauria de ser robusta en el sentit que el fet de processar la imatge (canviar la seva mida, retallar-la, invertir-la, comprimir-la, etc.) no ha d'impedir la recuperació de la marca.

- Quan la marca identifiqui el comprador, aquesta hauria de resistir atacs de confabulació de dos compradors. És a dir, si dos compradors adquireixen dues imatges amb dues marques diferents que els identifica, aquests no haurien de poder esborrar la part de la marca que justament diferencia un comprador de l'altre.

## 4. ENFOCAMENT I MÈTODE SEGUIT

El TFC consistirà bàsicament en el desenvolupament de dos programes: **marca**, un per marcar i extreure la marca d'una imatge, **confabula** per confabular dues (o més) imatges.

Els elements principals de què compte el programa **marca** són:

- Encastador de la marca d'aigua sobre una imatge: insereix una marca d'aigua en una imatge.

- Extractor de la marca d'aigua d'una imatge: llegeix la marca d'aigua d'una imatge marcada.

- Desconfabulador: a partir de dues imatges alterades per dos compradors confabuladors, permet recuperar la marca que identifiqui com a mínim un dels dos compradors.

Aquests tres elements interaccionen amb una petita base de dades que conté la identitat dels compradors / propietaris de les imatges.

El programa **confabula** permet, a partir de dues imatges comprades al mateix venedor per compradors diferents, alterar la imatge per esborrar-ne la identitat

Per provar el sistema s'ha dut a terme un seguit de jocs de prova per comprovar el funcionament correcte del programari així com per estudiar la seva robustesa.

L'entorn de programació escollit ha estat el llenguatge C sota Linux. No s'ha utilitzat cap llibreria excepte les pròpies del llenguatge C. Els programes s'executen a la línia de comandes.

## 5. PLANIFICACIÓ DEL PROJECTE

El projecte es descompon en les tasques següents:

- **T1**: cerca d'informació
- **T2**: disseny
- **T3**: implementació
- **T4**: proves
- **T5**: redacció de la memòria

En el quadre següent es desglossen cadascuna de les tasques en les seves subtasques:

| | |
|---|---|
| **T1.1** | Cerca d'informació sobre el format d'imatge BMP |
| **T1.2** | Cerca d'informació sobre protecció del copyright amb detecció de còpia usant tècniques de marca d'aigua. |
| **T1.3** | Cerca d'informació sobre codis de Hamming duals i el seu ús contra confabulacions |
| **T2.1** | Dissenyar un senzill mètode de marca d'aigua. |
| **T2.2** | Dissenyar l'algorisme que permetrà encastar una marca que identifiqui un comprador. |
| **T2.3** | Dissenyar l'algorisme que permetrà llegir la marca d'un BMP i obtenir la identitat del comprador. |
| **T2.4** | Dissenyar un programa que, donades dues imatges, en generi una a partir de la confabulació dels dos compradors. |
| **T3.1** | Implementació del mètode de marca d'aigua i prova sobre imatges. |
| **T3.2** | Implementació del programa d'encast / extracció de marca. |
| **T3.3** | Implementació del programa de confabulacions. |
| **T4.1** | Fer jocs de proves sobre robustesa del sistema. |
| **T4.2** | Fer jocs de proves sobre la resistència a confabulacions de 2 usuaris. |
| **T5.1** | Elaboració de la memòria: estat de l'art. |
| **T5.2** | Elaboració de la memòria: disseny del programa i algorismes. |
| **T5.3** | Elaboració de la memòria: breu manual d'usuari de l'aplicació. |
| **T5.4** | Elaboració de la memòria: apèndix amb codi font, degudament comentat. |
| **T5.5** | Fer presentació del projecte, amb PowerPoint o altres aplicacions. |

A partir del quadre anterior, s'ha seguit la següent planificació:

| Setmana | Dates | Activitat | Esdeveniment |
|---|---|---|---|
| 1 | 28 febrer | Definir els objectius del projecte | Trobada presencial |
| 2 | 1-7 març | T1.1 + T1.2 | |
| 3 | 8-14 març | T1.1 + T1.2 + T1.3 + Redacció Pla Treball | Lliurament del Pla de Treball |
| 4 | 15-21 març | T1.3 + T2.1 | |
| 5 | 22-28 març | T2.1 + T2.2 | |
| 6 | 29 març - 4 abril | T2.2 + T3.1 | |
| 7 | 5 - 11 abril | T2.3 + T3.2 | |
| 8 | 12 - 18 abril | T2.4 | Lliurament PAC 2 |
| 9 | 19 - 25 abril | T2.4+T3.3 | |
| 10 | 26 abril - 2 maig | T3.3 | |
| 11 | 3 - 9 maig | T4.1 + T4.2 | |
| 12 | 10 -16 maig | T4.1 + T4.2 | |
| 13 | 17 - 23 maig | T5.1 | Lliurament PAC 3 |
| 14 | 24 - 30 maig | T5.1 + T5.2 | |
| 15 | 31 maig - 6 juny | T5.3 + T5.4 | |
| 16 | 7 - 13 juny | T5.5 | Lliurament Pre-memòria |
| 17 | 14 - 20 juny | | Lliurament final |

## 6. DISSENY DEL SISTEMA

El disseny del sistema gira entorn de quatre algorismes:

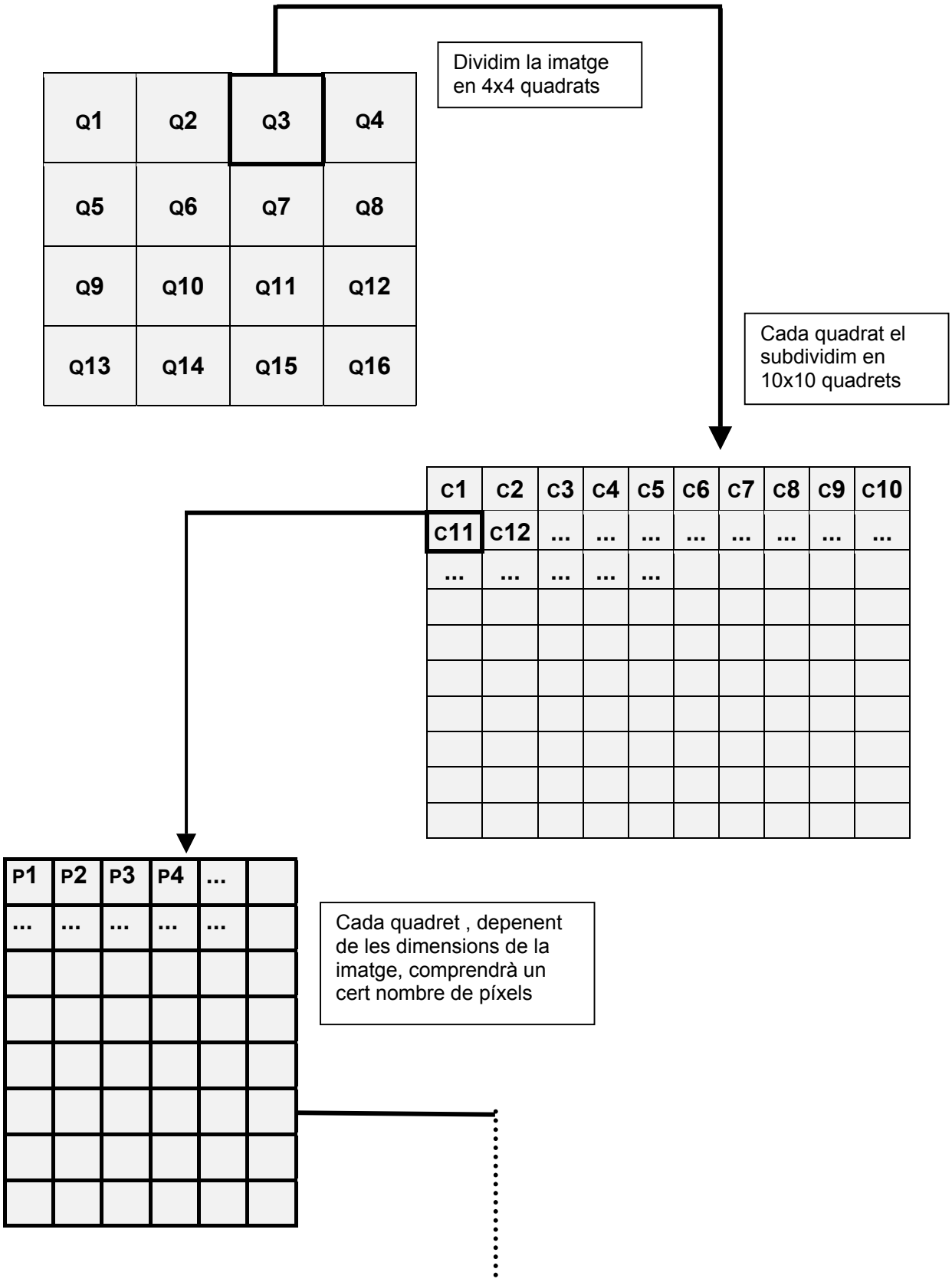
- algorisme d'encast de la marca
- algorisme d'extracció de la marca
- algorisme de confabulació
- algorisme anticonfabulador

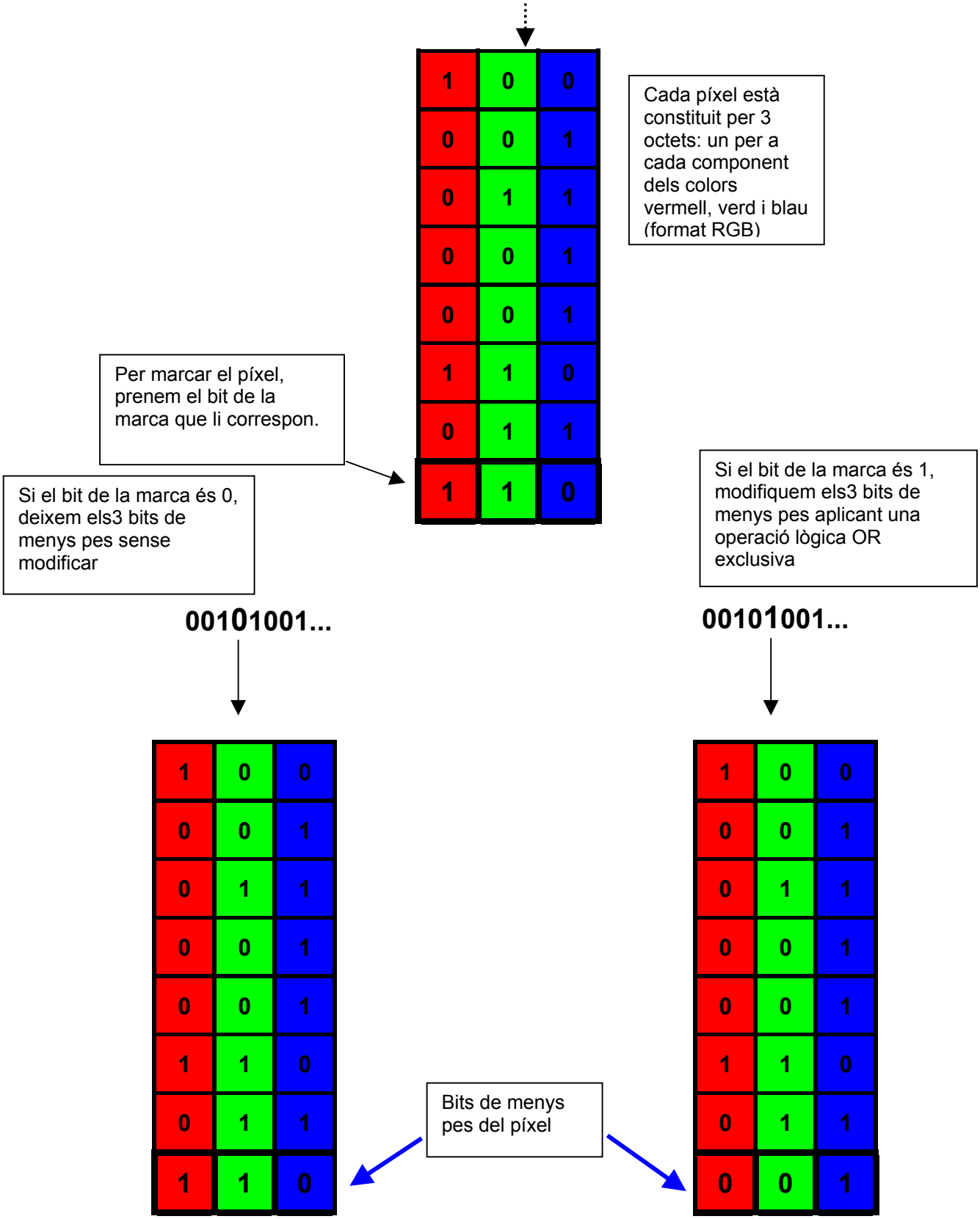
A continuació detallem les característiques de cadascun:

- **ALGORISME D'ENCAST DE LA MARCA**

Per encastar una marca en una imatge, subdividim la mateixa en 16 quadrats. (Recordem que treballarem sempre amb imatges quadrades). A continuació cada quadrat es subdivideix en 100 quadrets. Aleshores, cadascun d'aquests quadrets (que contindrà un cert nombre de píxels) es marcarà, de manera consecutiva amb els bits de la marca. Així, posem el valor del bit de menys pes de la marca al primer quadret, el segon bit de menys pes de la marca en el segon quadret, i així successivament. Quan hem arribat a l'últim bit de la marca (el de més pes), repetim el procés amb el bit de menys pes. Això ho fem per a cada quadrat de la imatge.

En les dues pàgines següents mostrem un esquema del procés de marcatge. Treballarem sempre amb imatges BMP quadrades, de 24 bits, sense comprimir i amb marques de 7 bits: 3 bits per identificar el compradors i 4 bits de correcció d'errors que constitueixen el codi dual de Hamming.


Q1
Q2
Q3
Q4
Q5
Q6
Q7
Q8
Q9
Q10
Q11
Q12
Q13
Q14
Q15
Q16
Dividim la imatge en 4x4 quadrats
Cada quadrat el subdividim en 10x10 quadrets
c1
c2
c3
c4
c5
c6
c7
c8
c9
c10
c11
c12
...
P1
P2
P3
P4
Cada quadret , depenent de les dimensions de la imatge, comprendrà un cert nombre de píxels


1 0 0
0 0 1
0 1 1
0 0 1
0 0 1
1 1 0
0 1 1
1 1 0
Cada píxel està constituit per 3 octets: un per a cada component dels colors vermell, verd i blau (format RGB)
Per marcar el píxel, prenem el bit de la marca que li correspon.
Si el bit de la marca és 0, deixem els3 bits de menys pes sense modificar
Si el bit de la marca és 1, modifiquem els3 bits de menys pes aplicant una operació lògica OR exclusiva
00101001...
00101001...
1 0 0
0 0 1
0 1 1
0 0 1
0 0 1
1 1 0
0 1 1
1 1 0
Bits de menys pes del píxel
1 0 0
0 0 1
0 1 1
0 0 1
0 0 1
1 1 0
0 1 1
0 0 1

- **ALGORISME D'EXTRACCIÓ DE LA MARCA**

Donada una imatge marcada, la subdividim en 16 quadres i cada quadre en 100 quadrets. Per esbrinar la marca fem un procés de votació. Mantenim dos vectors de vots, de dimensió igual a la mida de la marca: V0 i V1. Els elements $V0_i$ i $V1_i$ d'aquests vectors es van actualitzant de la manera següent.

Recorrem els quadres com toqui. Segons la posició on ens trobem, estarem esbrinant un determinat bit de la marca. Aleshores, si dos o tres dels bits de menor pes de cada octet que configuren un píxel difereixen dels corresponents de la imatge sense marcar, vol dir que el més probable és que el bit de la marca sigui un "1". En cas contrari suposarem que el bit de la marca és un "0". En el primer cas augmentarem en una unitat l'element $V1_i$ del vector V1, en el segon cas farem el mateix amb l'element $V0_i$ del vector V0.

Després de recórrer tota la imatge marcada es fa el recompte de vots, de manera que si $V0_i > V1_i$ tindrem $m_i = 0$; $m_i = 1$ en cas contrari.

A continuació, la marca així obtinguda es passa pel test de Hamming. En haver codificat les marques com a codis duals de Hamming, el test de Hamming permet corregir errors simples i detectar-ne de dobles. Si no hi ha hagut error, o aquest ha estat simple (és a dir, un dels bits és erroni), la correcció de Hamming permet recuperar la marca. Si l'error és doble, la correcció de Hamming retorna tres codis corregits, els tres que amb un error doble donen lloc al codi erroni. Aleshores, dels tres codis escollim aquell que es troba a menor distància dels vots obtinguts; és a dir, aquell que minimitza les diferències entre els vots obtinguts i els bits del codi corregit.

- **ALGORISME DE CONFABULACIÓ**

Per a confabular dues imatges iguals marcades per un mateix venedor, comparem píxel a píxel les imatges marcades i si el bit de menys pes d'algun dels tres octets que el defineixen és diferent, canviem aleatòriament el seu valor. D'aquesta manera aconseguim esborrar part de la marca. La imatge obtinguda per aquest procediment serà la imatges confabulada dels dos compradors. El procés així descrit es pot estendre a més de dos confabuladors. A més confabuladors, més bits de la marca poden quedar modificats i més difícil és la seva recuperació.

Un píxel de la imatge marcada del **comprador 1**

| | | |
|---|---|---|
| 1 | 0 | 0 |
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 0 | 0 | 1 |
| 0 | 0 | 1 |
| 1 | 1 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 0 |

Píxel corresponent de la imatge marcada del **comprador 2**

| | | |
|---|---|---|
| 1 | 0 | 0 |
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 0 | 0 | 1 |
| 0 | 0 | 1 |
| 1 | 1 | 0 |
| 0 | 1 | 1 |
| 0 | 0 | 0 |

Algun dels bits de menys pes del píxel són diferents. Canviem doncs aleatòriament els valors d'aquests bits. Escollim el valor d'un bit a l'atzar...

Si el bit és 0 el píxel confabulat resta de la manera següent:

| | | |
|---|---|---|
| 1 | 0 | 0 |
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 0 | 0 | 1 |
| 0 | 0 | 1 |
| 1 | 1 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 0 |

Si el bit és 1...

| | | |
|---|---|---|
| 1 | 0 | 0 |
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 0 | 0 | 1 |
| 0 | 0 | 1 |
| 1 | 1 | 0 |
| 0 | 1 | 1 |
| 0 | 1 | 1 |

- **ALGORISME ANTICONFABULADOR**

L'algorisme anticonfabulació es basa en la robustesa del codi dual de Hamming. En afegir bits de redundància a la tira de bits que identifica un comprador, podem corregir errors simples i detectar errors dobles. De fet, es pot demostrar que, en el cas de la confabulació de dos compradors, aquesta codificació permet recuperar la marca d'un dels dos confabuladors en tots els casos, sempre i quan no hi hagi hagut processament addicional de les imatges (*). En el cas de confabulació de tres només en alguns casos es pot recuperar la marca, tal i com es pot comprovar en el joc de proves que es detallen més endavant. A continuació detallem el funcionament dels codis correctors de Hamming.

Per codificar una paraula de 3 bits de manera que ens permeti corregir errors simples i detectar errors dobles, afegim als 3 bits d'informació 4 bits de paritat:

**x x x c1 c2 c3 p**

Si enumerem els bits d'esquerra a dreta i de 1 a 7, tenim que

**c1**: paritat dels bits 1, 2 i 3
**c2**: paritat dels bits 1 i 2
**c3**: paritat dels bits 1 i 3
**p**: paritat dels bits 1 a 6

Per exemple, si volem codificar **110** tindrem que **c1 = 0**, **c2 = 0**, **c3 = 1**, **p = 0**:

**110 001 1**

Per detectar errors, construïm un numero binari de 4 bits anomenat síndrome que conté els valors de paritat següents:

**s1 s2 s3 s4**

**s1**: paritat dels bits 1, 2, 3 i 4 del codi
**s2**: paritat dels bits 1, 2 i 5 del codi
**s3**: paritat dels bits 1, 3 i 6 del codi
**s4**: paritat de tots els bits del codi

És evident que si tots aquests bits són zero el codi és correcte. Si hi ha hagut algun error (simple o doble) el síndrome serà diferent de zero. Per exemple, un síndrome igual a **0111** voldrà dir que hi ha hagut un error simple en el segon bit del codi; un síndrome igual a **1100** voldrà dir que hi ha hagut error doble: o bé en els bits 3 i 5, o en els bits 2 i 6, o en els bits 1 i 7.

En aquest darrer cas, dels tres codis escollim aquell que es troba a menor distància dels vots obtinguts en el procés d'extracció de la marca; és a dir, aquell que minimitza les diferències entre els vots obtinguts i els bits del codi corregit.

## 7. IMPLEMENTACIÓ DEL SISTEMA

El sistema de marca d'aigua i anticonfabulació per a imatges BMP presentat en aquest TFC està implementat en el programa **marca**. L'algorisme de confabulació de dos o més compradors s'implementa en el programa **confabula**. A més, hem implementat dues rutines, **infobmp** i **cmpbmp**, que han resultat molt útils a l'hora de provar el sistema. La primera imprimeix els valors dels camps d'un fitxer BMP i la segona ens compara, píxel a píxel dues imatges BMP.

El programa **marca** consta dels fitxers següents:

marca.h
marca.c
pMarca.c
eMarca.c
hamMarca.c
bdMarca.c
errMarca.c
bmp.h
bmp.c

- **marca.h**

Fitxer de constants i capçaleres de funcions del sistema

- **marca.c**

Programa principal. Segons l'opció introduïda per l'usuari a la línia de comandes el programa encastarà o llegirà la marca.

- **pMarca.c**

Mòdul per encastar la marca a una imatge tal com es descriu en l'apartat de disseny.

- **eMarca.c**

Mòdul per extreure la marca d'una imatge. La funció retorna el nom del comprador i un valor numèric que expressa un tant per cent que indica el grau d'ajustament entre la marca i els vots obtinguts en el recompte dels bits de la marca. Un valor del 100% vol dir que tots els vots estan d'acord amb els bits de la marca, un 0% vol dir que s'ha obtingut empat de vots i que, per tant, la marca obtinguda no és fiable.

La funció te en compte que la imatge marcada hagi pogut estar retallada i/o invertida. Si la mida de la imatge marcada marcada es inferior a l'original, la funció busca el millor encaix de la primera dins de la segona tot comparant tots els bits de cada píxel excepte el bit inferior que, lògicament, haurà estat marcat. Si es troba l'encaix, la funció fa el recompte de vots ja que "sap" quin es el bit de la marca corresponent a cada píxel.

El mòdul inclou les funcions:

**fitBmp**: troba l'encaix d'una imatge retallada dins de la imatge original
**pesCodi**: calcula el pes d'un codi de Hamming en relació als vots obtinguts
**votaTota**: fa el procés de votació d'una imatge sencera
**votaRetall**: fa el procés de votació d'una imatge retallada

- **hamMarca.c**

Implementa la correcció dels codis duals de Hamming. La funció retorna un valor numèric que ens indica si el codi passat a la funció com a paràmetre presenta 0, 1 o 2 errors. En cas d'error simple, la funció retorna també per referència el codi corregit, si l'error és doble, la funció retorna un vector de tres elements corresponents a les possibles correccions del codi.

El mòdul inclou la funció:

**paritat**: calcula la paritat d'un codi

- **bdMarca.c**

Mòdul que interacciona amb la base de dades dels clients. Permet trobar la clau corresponent a un client de la base de dades o bé, donada una clau, cercar el client a qui pertany.

El mòdul inclou les funcions:

**getMarca**: donat el nom d'un client, cerca a la base de dades la marca que li correspon

**getClient**: donada una marca, cerca a la base de dades el nom del client que li correspon

- **errMarca.c**

Mòdul que gestiona els errors i/o excepcions del programa. Els missatges d'errors i els seus codis estan definits al fitxer marca.h.

- **bmp.h**

Fitxer de constants i capçaleres de funcions per a la gestió d'operacions sobre fitxers BMP.

- **bmp.c**

Mòdul que gestiona les operacions sobre fitxers BMP, com ara la seva càrrega en memòria, còpia a disc, control dels camps de les capçaleres, lectura i escriptura dels camps de les capçaleres, inversió horitzontal i vertical de les dades de la imatge, gestió d'errors.

El mòdul inclou les funcions:

**getValor**: llegeix un valor enter en un lloc de memòria (útil per accedir als camps de les capçaleres d'un fitxer BMP carregat a la memòria)

**setValor**: escriu un valor enter en un lloc de memòria (útil per modificar els camps de les capçaleres d'un fitxer BMP carregat a la memòria)

**loadBmp**: carrega una imatge BMP a la memòria

**chkBmp**: comprova alguns camps de les capçaleres d'una imatge BMP carregada a la memòria. En concret, comprova que el format és BMP, que la imatge és de 24 bits, quadrada i sense comprimir.

**saveBmp**: guarda al disc una imatge BMP

**invhBmp**: inverteix horitzontalment els píxels d'una imatge BMP

**invvBmp**: inverteix verticalment els píxels d'una imatge BMP

**errBmp**: gestiona els errors i/o excepcions relacionats amb la gestió de fitxers BMP.

El programa **confabula** consta d'aquests fitxers:

marca.h
confabula.c
errMarca.c
bmp.h
bmp.c

- **confabula.c**

Implementa la confabulació de dos o més compradors tal i com s'ha descrit en l'apartat de disseny.

El mòdul inclou la funció:

**conf**: fa pròpiament la confabulació de les imatges

La resta de fitxers són els mateixos que els del programa **marca**

El programa **infobmp** consta d'aquests fitxers:

bmp.h
infobmp.c
bmp.c

El mòdul inclou la funció:

**infoBmp**: Imprimeix els continguts de les capçaleres i del camp de dades (en decimal) d'un fitxer BMP.

El programa **cmpbmp** consta d'aquests fitxers:

bmp.h
cmpbmp.c
bmp.c

El mòdul inclou la funció:

**cmpBmp**: Imprimeix el nombre d'octets i el nombre de píxels en què difereixen dues imatges BMP de la mateixa mida i els percentatges respecte del total.

## 8. INSTAL·LACIÓ DEL PROGRAMARI

Copieu el fitxer arxivador **tfc.tar** en una carpeta del linux com ara **usr/local**. A continuació fer a la linia de comandes:

**%> tar xvf tfc.tar**

Es crearà el directori tfc amb els fitxers font, la base de dades **marca.dat**, el *script* **confabula.sh** i el **Makefile**. A continuació fer:

**%> make**

Es crearan els binaris **marca, confabula, infobmp** i **cmpbmp**.

## 9. MANUAL D'USUARI

- **MARCA. PER ENCASTAR LA MARCA**

Per a marcar una imatge, a la línia de comandes del linux fer:

**%> marca imatge.bmp m comprador**

on **imatge.bmp** és una imatge de tipus bmp, de 24 bits, quadrada i sense comprimir i **comprador** és el nom d'un comprador que ha d'estar en el fitxer base de dades **marca.dat.**

El programa crea un fitxer **imatge.comprador.bmp** amb la marca associada al comprador

Exemple d'execució del programa:

```
%> marca lenna.bmp m jordi
%>
%> ls -t *.bmp
%> lenna.jordi.bmp
%> lenna.bmp
```

- **MARCA. PER EXTREURE LA MARCA**

Per a extreure la marca d'una imatge, a la línia de comandes del linux fer:

**%> marca imatge.comprador.bmp e imatge.bmp**

on **imatge.comprador.bmp** és una imatge de tipus bmp, de 24 bits, quadrada, sense comprimir i prèviament marcada pel venedor i **imatge.bmp** és la imatge original sense marcar.

El programa "intentarà" esbrinar el nom del comprador i l'imprimirà a la pantalla juntament amb un percentatge de fiabilitat.

Exemples d'execució del programa:

Extraiem la marca d'una imatge marcada i no processada:

```
%> marca lenna.jordi.bmp e lenna.bmp
%> Comprador: jordi (100.00%)
```

Extraiem la marca d'una imatge confabulada per dos compradors:

```
%> marca lenna.conf.jordi.anna.bmp e lenna.bmp
%> Comprador: anna (43.03%)
```

Extraiem la marca d'una imatge marcada i processada (hem afegit un 10% de soroll aleatori):

```
%> marca lenna.jordi.soroll10percent.bmp e lenna.bmp
%> Comprador: jordi (12.21%)
```

Intentem extreure la marca d'una imatge confabulada per tres compradors:

```
%> marca lenna.conf.jordi.anna.claudia.bmp e lenna.bmp
%> Comprador no reconegut
```

- **INFOBMP**

## %> imfobmp imatge.bmp [> fitxer]

Exemple d'execució del programa:

```
%> imfobmp lenna.bmp > lenna.inf
%> cat  lenna.inf

NOM:
lenna.bmp

CAP:
bfType:                 19778
bfSize:                 480054
bfReserved1:            0
bfReserved2:            0
bfOffBits:              54

CAP INFORMACIO:
biSize:                 40
biWidth:                400
biHeight:               400
biPlanes:               1
biBitCount:             24
biCompression:          0
biSizeImage:            480000
biXPelsPerMeter:        2834
biYPelsPerMeter:        2834
biClrUsed:              0
biClrImportant:         0

DADES:
30 28 40-31 29 41-33 31 43-29 27 39-23 21 40-28 25 40-35 34 43-32 32 38-...
32 30 42-32 30 42-33 31 43-29 27 39-39 37 57-36 33 49-33 32 42-30 30 36-...
27 29 39-26 31 40-30 35 44-30 35 44-22 28 39-30 35 44-23 24 34-35 31 43-...
28 30 40-25 27 37-25 30 39-24 29 38-27 31 42-29 31 41-28 29 39-38 34 45-...
30 31 41-27 29 39-29 31 41-28 30 40-35 32 47-30 28 40-32 29 38-29 26 35-...
28 29 39-28 29 39-31 32 42-30 31 41-42 37 52-36 32 44-34 31 40-38 36 42-...
29 28 38-27 26 36-29 28 38-27 26 36-30 25 40-31 27 39-31 28 37-34 34 40-...
31 30 40-31 27 38-32 28 39-31 27 38-35 35 49-36 36 48-36 38 46-30 32 40-...
...
```

Observem que el camp de dades agrupa els octets per píxels (un per a cada component RGB) i els píxels en files que es corresponen a les files de píxel de la imatge però invertides verticalment.

- **CMPBMP**

## %> cmpbmp imatge1.bmp imatge2.bmp [> fitxer]

Exemple d'execució del programa:

```
%> cmpbmp lenna.bmp lenna.jordi.bmp
Les imatges difereixen en: 273600 octets (57.00%), 91200 pixels (57.00%)
```

- **CONFABULA**

## %> confabula imatge.comprador1.bmp imatge.comprador2.bmp ...

El programa crea un fitxer **imatge.conf.comprador1.comprador2.[...].bmp** amb la marca confabulada

Exemple d'execució del programa:

```
%> confabula lenna.jordi.bmp lenna.anna.bmp
%>
%> ls -t *.bmp
%> lenna.conf.jordi.anna.bmp
%> lenna.anna.bmp
%> lenna.jordi.bmp
%> lenna.bmp
%>
%> marca lenna.conf.jordi.anna.bmp e lenna.bmp
%> Comprador: anna (43.03%)
```

- **PER FER EL JOC DE PROVES DE LES CONFABULACIONS**

## %> confabula.sh [> fitxer]

El *script* fa un joc de proves de confabulacions. En total marca la imatge lenna.bmp i la imatge logo.bmp per a 7 compradors. A continuació fa un total de 10 confabulacions entre parelles de compradors i repeteix el procés 100 vegades (1000 confabulacions entre 2 compradors en total). Després el *script* fa 8 confabulacions de les dues imatges entre 3 compradors, també repetint el procés 100 vegades (800 confabulacions entre 3 compradors en total). Per a visualitzar els resultats és recomanable redireccionar la sortida a un fitxer.

Exemple d'execució del *script* i d'un fragment del fitxer joc de proves resultant:

```
%> confabula.sh>confabula.inf
%> cat confabula.inf

Script Joc de Proves Confabulacions
TFC (Proteccio del copyright)
Jordi Trubat Dalmau, jtrubat@uoc.edu

ENCAST I EXTRACCIO DE LA MARCA PER A CLIENTS (IMATGE: lenna)

> Marcant lenna.bmp per al client jordi...
Fitxer creat: lenna.jordi.bmp
> Extraient la marca de lenna.jordi.bmp...
Comprador: jordi (100.00%)

> Marcant lenna.bmp per al client anna...
Fitxer creat: lenna.anna.bmp
> Extraient la marca de lenna.anna.bmp...
Comprador: anna (100.00%)

> Marcant lenna.bmp per al client claudia...
Fitxer creat: lenna.claudia.bmp
> Extraient la marca de lenna.claudia.bmp...
Comprador: claudia (100.00%)

...

CONFABULACIONS DE 2 COMPRADORS I EXTRACCIO DE LA MARCA (IMATGE: lenna)

> Creant 100 confabulacions de jordi i anna i llegint la marca...

Comprador: jordi (43.00%)
Comprador: anna (43.09%)
Comprador: anna (43.09%)
Comprador: jordi (43.21%)
Comprador: jordi (43.21%)
Comprador: jordi (43.08%)
Comprador: jordi (43.08%)
Comprador: jordi (43.08%)
Comprador: jordi (43.18%)
Comprador: jordi (43.18%)
Comprador: anna (43.03%)
Comprador: anna (43.03%)
Comprador: anna (43.03%)
Comprador: anna (43.25%)
Comprador: anna (43.25%)
Comprador: jordi (43.00%)
...
```

## 10. JOC DE PROVES

Per fer els jocs de proves s'ha treballat bàsicament amb la imatge lenna.bmp de dimensions 400x400 i la imatge logo.bmp que és originalment una imatge en format gif guardada en format bmp.

El programa **marca** retorna la marca correcta d'una comprador en els casos següents:

- Quan la imatge marcada no s'ha sotmès a cap processament addicional
- Quan la imatge marcada ha estat retallada i no s'ha sotmès a cap processament addicional
- Quan la imatge ha estat invertida horitzontalment, verticalment o horitzontalment i verticalment (doble reflexió).
- Quan la imatge ha estat retallada i invertida horitzontalment, verticalment o horitzontalment i verticalment no s'ha sotmès a cap processament addicional
- Quan hi ha hagut confabulació de dos compradors
- Quan a la imatge se li afegit soroll aleatori imperceptible
- Quan a la imatge se li afegit un filtre de Gauss prou baix
- En el cas de logo.bmp, quan se l'ha comprimit amb una qualitat del 90%

En el cas de les confabulacions, el *script* confabula.sh serveix per fer un joc de proves de confabulació. En aquest joc de proves es marca la imatge lenna.bmp i la imatge logo.bmp per a 7 compradors. A continuació fa un total de 10 confabulacions entre parelles de compradors i repeteix el procés 100 vegades (1000 confabulacions entre 2 compradors en total). En tots els casos es recupera la marca d'un dels dos compradors. Després el *script* fa 8 confabulacions de les dues imatges entre 3 compradors, també repetint el procés 100 vegades (800 confabulacions entre 3 compradors en total). Una execució típica del script mostra que en aquest cas només al voltant del 60% dels casos es recupera la marca d'un dels tres compradors.

Per tant, l'ús dels codis duals de Hamming permeten sempre recuperar la marca d'un dels dos client confabuladors sempre que la imatge no hagi estat sotmesa a cap processament addicional, però no quan la confabulació té lloc entre 3 o més compradors.

Els resultats dels jocs de proves no són tant satisfactoris en els dos casos següents:

- Quan la imatge ha estat escalada
- Pel cas de *lenna.bmp*, comprimida a format JPEG

Els dos casos estan relacionats: quan es comprimeix una imatge a format JPEG o s'escala, es crea una nova imatge amb uns nous píxels, els valors dels quals, en general, són combinació dels valors dels píxels veïns de la imatge original. En aquest procés, els valors originals dels píxels es perden i, per tant, també es perden els bits de la marca. Només en el cas que la imatge presenti poc detall (com és el cas de la imatge logo.bmp) i la compressió sigui moderada, es pot recuperar la marca.

En el cas de l'escalatge, hem de tenir en compte que els algorismes d'escalatge més utilitzats són:

- Duplicació/eliminació de píxels
- Interpolació bilineal
- Interpolació bicúbica.

El primer mètode és el més senzill però el que dona pitjors resultats. S'utilitza poc per escalar imatges. Quan un programa de processament d'imatges utilitza un algorisme d'escalatge el més probable és que utilitzi el mètode de la interpolació bilineal, que dona resultats prou bons en relació al cost d'execució de l'algorisme.

La interpolació bilineal consisteix en crear nous píxels (tant si es redueix com si s'augmenta la imatge) que són valors interpolats dels valors dels píxels de la imatge original que es troben al seu voltant. La interpolació bicúbica es basa en el mateix principi però prenent més píxels per fer la interpolació però a un cost més alt d'execució.

Només podrem recuperar la marca en el cas que la imatge hagi estat escalada pel mètode de duplicació/eliminació de píxels o escalada en un múltiple de 100% de la mida original utilitzant els altre mètodes (en aquest cas entre els píxels originals s'insereixen de nous deixant inalterats els originals).

Resumim els resultats dels jocs de proves en la taula següent:

| Tipus de processament | Es recupera la marca? | Observacions |
|---|---|---|
| Imatge només marcada | Sí | Sense cap processament addicional |
| Imatge invertida horitzontalment | Sí | |
| Imatge invertida verticalment | Sí | |
| Imatge invertida horitzontal i verticalment | Sí | |
| Porció de la imatge marcada | Sí | Sempre i quan no hi hagi cap mena de processament addicional |
| Porció de la imatge invertida horitzontalment | Sí | Sempre i quan no hi hagi cap mena de processament addicional |
| Porció de la imatge invertida verticalment | Sí | Sempre i quan no hi hagi cap mena de processament addicional |
| Porció de la imatge invertida horitzontalment | Sí | Sempre i quan no hi hagi cap mena de processament addicional |
| Porció de la imatge invertida horitzontal i verticalment | Sí | Sempre i quan no hi hagi cap mena de processament addicional |
| Addició de soroll aleatori (10%) | Sí | Observem que a aquest nivell la degradació de la imatge és perceptible |
| Addició de soroll uniforme | No | |
| Filtre de Gauss (1%) | SÍ | |
| Escalatge bilineal | No | |
| Compressió JPEG (10%) | Sí | En el cas de logo.bmp |
| Compressió JPEG (40%) | No | |
| Confabulació de 2 compradors | Sí | Sempre que no hi hagi processament addicional |
| Confabulació de 3 compradors | Sí/No | Sí en 60% dels casos |

## 11. CONCLUSIONS

En aquest TFC hem implementat un senzill sistema de protecció del copyright per a imatges BMP mitjançant detecció de còpia amb utilització de marques d'aigua. El procés de marcatge d'una imatge ha consistit en modificar els valors dels bits de menys pes dels tres octets d'alguns píxels de la imatge en funció d'un codi (tira de bits) que identifica el comprador de la imatge. El codi utilitzat és un codi dual de Hamming que el fa més robust a possibles modificacions posteriors de la imatge i, en concret, permet la recuperació de la marca en tots els casos quan hi ha hagut confabulació de dos compradors.

La implementació de l'algorisme d'extracció de marca permet llegir la marca sense problemes quan la imatge marcada ha estat retallada (sense processament addicional) i/o invertida. A més d'aquests casos, l'algorisme permet recuperar la marca quan a la imatge se li ha afegit un cert nivell de soroll aleatori, se li ha aplicat un filtre de Gauss prou petit, en alguns casos de compressió al format JPEG, en tots els casos de confabulació de dos compradors i en un 60% dels casos en el cas de confabulació de tres compradors.

En general, els casos en què la imatge ha estat escalada l'algorisme d'extracció de marca no dona bons resultats, ni tampoc en els casos de compressió de la imatge a format JPEG quan aquesta és relativament rica en detalls o el nivell de qualitat de la compressió és baix.

La millora dels sistema de marcatge estudiat en aquest TFC passaria, doncs, per millorar la robustesa del sistema en els casos que hem apuntat abans i ampliar l'estudi a altres tipus d'atacs que no hem tingut en compte en aquest estudi (rotacions, deformacions, soroll uniforme, etc.) així com ampliar la robustesa del sistema en el cas de confabulacions de més de dos compradors.

## 12. GLOSSARI

**BMP :** Format de fitxer d'imatges introduït originàriament pel sistema operatiu Windows. Les imatges BMP poden assignar des d'un octet a cada píxel (imatges en blanc i negre) fins a 3 octets per píxel (16.7 milions de colors). Encara que hi ha la possibilitat de comprimir-les, rara vegada s'usa a la pràctica.

**Codi anticonfabulació:** Codi robust a la confabulació (normalment de dos compradors) d'una imatge marcada. És a dir, un codi que permet recuperar la marca en tots els casos en què la imatge no ha estat sotmesa a cap processament addicional.

**Codi dual de Hamming:** Codi que consisteix en afegir bits de detecció d'error al codi que identifica un comprador i que permet corregir errors simples de bit i detectar errors dobles.

**Confabulació:** Quan dos o més compradors d'una mateixa imatge comprada a un mateix venedor comparen un a un els octets de les seves imatges i els modifiquen amb la intenció d'esborrar la identitat (és a dir, part dels bits de la marca) dels mateixos.

**Encastar la marca:** Procés mitjançant enterrem en una imatge la identitat del comprador; per exemple, modificant els valors d'alguns bits de la mateixa.

**Extreure la marca:** Procés mitjançant recuperem la identitat del comprador d'una imatge prèviament marcada

**JPEG:** (*Joint Photographic Experts Group*) Mecanisme estàndard de compressió d'imatges de color o de escala de grisos amb reducció de qualitat.

**Marca:** Tira de bits que identifica el comprador d'una imatge en el cas del *fingerprinting* o del venedor en el cas del *watermarking*.

**Píxel:** (*Picture element*) Punt mínim de color visible d'una imatge. En el cas de les imatges BMP de 24 bits, tres octets, un per a cada component de color (vermell, verd, blau) defineixen un píxel.

**RGB:** (*Red, green, blue*) Sigles que identifiquen els tres components bàsics d'un color.

## 13. REFERÈNCIES

The .bmp file format

http://www.fortunecity.com/skyscraper/windows/364/bmpffrmt.html

BMP image format

http://astronomy.swin.edu.au/~pbourke/dataformats/bmp/

Digital WatermarkingWorld

http://www.watermarkingworld.org/

Web Reference - Watermarks

http://www.webreference.com/multimedia/watermarks.html

WWW References on Multimedia Watermarking and Data Hiding Research & Technology

http://www-nt.e-technik.uni-erlangen.de/~hartung/watermarkinglinks.html

Steganography & Digital Watermarking - Information Hiding -

http://www.jjtc.com/Steganography/

Digital Watermarking

http://www.cosy.sbg.ac.at/~pmeerw/Watermarking/

Safeguarding Digital Library Contents and Users. Digital Watermarking

http://www.alpvision.com/watermarking.html

Digital Watermarking

http://ei.cs.vt.edu/~h3004fox/f97/class/dw.html

Watermarking and Steganography Links

http://ee.tamu.edu/~deepa/wtrmk.html

Watermarking Research

http://www.gts.tsc.uvigo.es/gpsc/wmark/

Simple Collusion-Secure Fingerprinting Schemes for Images

http://csdl.computer.org/comp/proceedings/itcc/2000/0540/00/05400128abs.htm

The Laws of Cryptography:The Hamming Code for Error Correction

http://www.cs.utsa.edu/~wagner/laws/hamming.html

Robust Mesh Watermarking

http://www.cs.princeton.edu/gfx/proj/meshwm/meshwm.pdf

**Feynman Lectures on Computation**. Richard P. Feynman. Penguin Books

## 13. ANNEX. LLISTATS DELS FITXERS FONT

```
/*****************************************************************************
 marca.h

 TREBALL FI DE CARRERA - PROTECCIO DEL COPYRIGHT

 Jordi Trubat Dalmau, jtrubat@uoc.edu
*****************************************************************************/

/* Nom de la base de dades */
#define ARXIU_DADES "marca.dat"

/* Fixem a 7 bits la mida de la marca */
#define MIDA_MARCA 7

/* Opcions del programa */
#define OPCIO_ENCASTAR 1
#define OPCIO_EXTREURE 2

/* Nombre maxim de confabuladors */
#define MAX_CONFS 4

/* Marge minim de confiança (en tant per %) per a la marca */
#define MIN_MARGE 1.0

/* Codis d'error i missatges */
#define E_MAR_NPARAMETRESINC 500
#define E_MAR_NPARAMETRESINC_M "Nombre de parametres incorrecte\n"
#define E_MAR_PARAMETREINC 501
#define E_MAR_PARAMETREINC_M "Parametre incorrecte\n"
#define E_MAR_EXTINC 502
#define E_MAR_EXTINC_M "Extensio de fitxer incorrecta\n"
#define E_MAR_CLIENTINC 503
#define E_MAR_CLIENTINC_M "No existeix aquest comprador\n"
#define E_MAR_IMATGESDIF 504
#define E_MAR_IMATGESDIF_M "Imatges diferents\n"
#define E_MAR_NOMEM 505
#define E_MAR_NOMEM_M "No hi ha prou memoria\n"
#define E_MAR_FTXINOBRIR 506
#define E_MAR_FTXINOBRIR_M "No s'ha pogut obrir la base de dades\n"
#define E_MAR_FTXINTANCAR 507
#define E_MAR_FTXINTANCAR_M "No s'ha pogut tancar la base de dades\n"
#define E_MAR_MASSACONFS 508
#define E_MAR_MASSACONFS_M "Massa confabuladors\n"
#define E_MAR_DESC_M "Error desconegut :-(\n"


/* Definicio de tipus MARCA */
typedef unsigned int marca;

/*** Funcions ***/

/* Encasta la marca */
int pMarca(bmp *, marca);
```

```
/* Extreu la marca */
char *eMarca(bmp, bmp, float *);

/* Correccio de la marca */
int hamMarca(unsigned int, unsigned int *);

/* Cerca la marca d'un client a la base de dades */
int getMarca(char *, char *, marca *);

/* Cerca un client corresponent a una marca a la base de dades */
char *getClient(char *, marca);

/* Imprimeix un missatge d'error */
int errMarca(int);



/*****************************************************************************
 marca.c

 TREBALL FI DE CARRERA - PROTECCIO DEL COPYRIGHT

 Jordi Trubat Dalmau, jtrubat@uoc.edu
*****************************************************************************/

#include<stdio.h>
#include<sys/stat.h>
#include<string.h>
#include<stdlib.h>
#include<bmp.h>
#include "marca.h"


/*****************************************************************************
 nom:

        marca - Encasta i/o llegeix una marca d'una imatge BMP

 sinopsi:

        marca imatgeOriginal m comprador
        marca imatgeMarcada e imatgeOriginal

 descripcio:

        Segons el valor del segon parametre passat a la linia de comandes, el
        programa encasta una marca o llegeix la marca d'una imatge.

        Si el parametre es 'm', s'encasta la marca corresponent al comprador
        (tercer   parametre) a la imatge (primer parametre). La relacio entre



        marca i  comprador es troba en el fitxer marca.dat.

        Si el parametre es 'e', el programa intenta llegir la marca de la
```

```
        imatge marcada (primer parametre) tot comparant-la amb la imatge
        original (tercer parametre). El programa escriu el nom del comprador
        i un tant per cent que indica el grau d'ajustament de la marca als
        vots obtinguts en el recompte dels bits de la marca.
*************************************************************************/

int main(int argc, char **argv)
{

        char *f1,*f2;           /* Nom del fitxer BMP */
        unsigned long mida;     /* Mida del fitxer BMP */
        octet *buf1, *buf2;     /* Punter al buffer on es guarda el BMP */
        char *client;           /* Nom del client */
        bmp img1, img2;         /* Instancia de l'estructura de dades BMP */
        marca m;                /* Marca */
        char *c;                /* Buffer pel nom del fitxer BMP marcat */
        float marge;            /* Bondat de la marca */
        int opcio;              /* Marcar o exterure? */

        /* Nombre de parametres correcte? */
        if(argc!=4)  {
                errMarca(E_MAR_NPARAMETRESINC);
                return -1;
        }

        /* El segon parametre ha de ser 'm' o 'e' */
        if(strcmp(argv[2],"m")==0)  {
                opcio=OPCIO_ENCASTAR;
        } else {
                if(strcmp(argv[2],"e")==0)  {
                        opcio=OPCIO_EXTREURE;
                } else {
                        errMarca(E_MAR_PARAMETREINC);
                        return -1;
                }
        }

        /* Opcio escollida */
        switch(opcio)
        {

        /***************************************
         * Volem encastar una marca a la imatge
         ***************************************/

        case OPCIO_ENCASTAR:

                /* Nom de la imatge BMP */
                f1=argv[1];
                if((strstr(f1,"bmp")==NULL)&&(strstr(f1,"BMP")==NULL)) {
                        errMarca(E_MAR_EXTINC);
                        return -1;
                }



                /* Nom del client */
                client=argv[3];
```

```
        /* Cerquem a la base de dades la marca del client */
        if(getMarca(ARXIU_DADES, client, &m)!=0)  {
                return -1;
        }

        /* Montem la cadena del BMP marcat */
        if((c=(char*)malloc(strlen(f1)+strlen(client)+1+1))==NULL)  {
                errMarca(E_MAR_NOMEM);
                return -1;
        }
        f2=strcpy(c, f1);
        f2=strcpy(c+strlen(f1)-3, argv[3]);
        f2=strcat(c,".bmp");

        /* Carreguem el BMP a la memoria */
        if((buf1=loadBmp(&mida, f1))==NULL)  {
                return -1;
        }

        /* Omplim els camps de l'estructura BMP */
        img1.nom=f1;
        img1.buf=buf1;

        /* Llegim l'estructura del BMP */
        if(chkBmp(img1)!=0)  {
                return -1;
        }

        /* Marquem el BMP */
        pMarca(&img1, m);

        /* Canviem el nom del BMP */
        img1.nom=f2;

        /* Guardem el BMP marcat */
        if(saveBmp(img1)!=0)  {
                return -1;
        }
        break;


/*************************************
 * Volem extreure la marca d'una imatge
 *************************************/

case OPCIO_EXTREURE:

        /* Nom de la imatge BMP marcada */
        f1=argv[1];
        if((strstr(f1,"bmp")==NULL)&&(strstr(f1,"BMP")==NULL)) {
                errMarca(E_MAR_EXTINC);
                return -1;
        }
```

```
		/* Nom de la imatge BMP original */
		f2=argv[3];
		if((strstr(f2,"bmp")==NULL)&&(strstr(f2,"BMP")==NULL)) {
			errMarca(E_MAR_EXTINC);
			return -1;
		}

		/* Carreguem el BMP marcat a la memoria */
		if((buf1=loadBmp(&mida, f1))==NULL)  {
			return -1;
		}

		/* Omplim els camps de l'estructura del BMP marcat*/
		img1.nom=f1;
		img1.buf=buf1;

		/* Llegim l'estructura del BMP marcat */
		if(chkBmp(img1)!=0)  {
			return -1;
		}

		/* Carreguem el BMP original a la memoria */
		if((buf2=loadBmp(&mida, f2))==NULL)  {
			return -1;
		}

		/* Omplim els camps de l'estructura del BMP original*/
		img2.nom=f2;
		img2.buf=buf2;

		/* Llegim l'estructura del BMP original */
		if(chkBmp(img2)!=0)  {
			return -1;
		}

		/* Llegim la marca encastada*/
		if((client=eMarca(img1,img2,&marge))==NULL)  {
			printf("Comprador no reconegut\n");
			break;

		}  else  {

			/* Imprimim el nom del comprador i la qualitat de la marca */
			printf("Comprador: %s (%.2f%%)\n", client, marge);
			break;

		}

	}

	/* Fet */
	return 0;

}
```

```
/*****************************************************************************
 pMarca.c

 TREBALL FI DE CARRERA - PROTECCIO DEL COPYRIGHT

 Jordi Trubat Dalmau, jtrubat@uoc.edu
*****************************************************************************/

#include<stdio.h>
#include<bmp.h>
#include "marca.h"



/*****************************************************************************
 pMarca

 Marca una imatge BMP amb la marca d'un client

 descripcio:

        El metode per encastar la marca es el que esta explicat al document
        "TFC. Proteccio del copyright". Es subdivideix la imatge original
        en 16 quadrats i cada quadrat en 100 quadrets. Aleshores per a cada
        quadret de pixels es modifica successivament el bit de menys pes de
        cada octet fen un OR exclusiu amb el bit de la marca. El sentit en
        que la rutina fa l'encast es de quadre en quadre, per a cada quadre
        de quadret en quadret, per a cada quadret, de pixel en pixel.

 entrada:

        img:     punter a una instancia de tipus BMP
        marca:  punter a una instancia de tipus MARCA

 sortida:

        img:     punter a una instancia de tipus BMP amb les dades marcades

 retorna:

        0
*****************************************************************************/

int pMarca(bmp *img, marca m)
{

        octet *b;               /* On comencen les dades del BMP */
        unsigned int w;         /* Amplada (altura) de la imatge en pixels */
        int pixQ;               /* Pixels per costat de cada quadrat */
        int pixC;               /* Pixels per costat de cada quadret */
        int x,y;                /* Coordenades en pixels d'un quadrat */
        int qi,qj;              /* Quadrat de la fila i columna j */
        int ci,cj;              /* Quadret de la fila i columna j */
        int iByt,jByt;          /* Octet ij dins d'un quadret */
        int iBit;               /* Numero de bit de la marca */
        unsigned int temp;      /* Copia de la tira de bits de la marca */
        int bit;                /* Bit de la marca */
        int fil, col;           /* Fila i columna auxiliars */
```

```
/* Buffer del BMP */
b=img->buf;

/* Amplada (altura) de la imatge en pixels */
w=getValor(BIWIDTH_S, b+BIWIDTH);

/* Subividim cada costat en 4 quadrats */
pixQ=w>>2;

/* Subividim cada quadrat en 10 quadrets */
pixC=pixQ/10;

/* Les dades del BMP comencen despres de les capcaleres */
b+=getValor(BFOFFBITS_S, b+BFOFFBITS);

/* Per a cada filera de quadrats... */
for(qi=0;qi<4;qi++)   {

  /* Per a cada columna de quadrats... */
  for(qj=0;qj<4;qj++)   {

          /* Comencem a marcar cada quadre amb el primer bit de la marca */
          iBit=0;

          /* Fem una copia de la marca */
          temp=m;

          /* Per a cada filera de quadrets... */
          for(ci=0;ci<10;ci++)   {

            /* Cada fila comenca en aquest pixel */
    x=qi*pixQ+ci*pixC;

        /* Guardem aquesta fila */
        fil=x;

        /* Per a cada columna de quadrets... */
            for(cj=0;cj<10;cj++)   {

                    /* Cada columna comenca en aquest pixel */
                    y=qj*pixQ*3+cj*pixC*3;

                    /* Guardem aquesta columna */
                    col=y;

                    /* Seleccionem el bit de la marca */
                    bit=temp&1;

                    /* Per a cada fila de pixels del quadret...*/
                    for(iByt=0;iByt<pixC;iByt++)  {

                      /* Per a cada columna de pixels del quadret...*/
                      for(jByt=0;jByt<pixC;jByt++)  {
```

```
                        /* Marquem els 3 octets */

                        /* Octet R */
                        b[x*w*3+y]=b[x*w*3+y]^bit;
                        y++;

                        /* Octet G */
                        b[x*w*3+y]=b[x*w*3+y]^bit;
                        y++;

                        /* Octet B */
                        b[x*w*3+y]=b[x*w*3+y]^bit;
                        y++;
                    }

                    /* Seguent fila de pixels */
                    x++;

                    /* Restituim la columna */
                    y=col;

                }

                /* Restituim la fila */
                x=fil;

                /* Seguent bit de la marca */
                if(iBit==MIDA_MARCA-1)  {
                        iBit=0;
                        temp=m;
                } else  {
                        iBit++;
                        temp=temp>>1;
                }

            }
        }
    }
}

/* Fet */
return 0;

}
```

```
/*****************************************************************************
 eMarca.c

 TREBALL FI DE CARRERA - PROTECCIO DEL COPYRIGHT

 Jordi Trubat Dalmau, jtrubat@uoc.edu
*****************************************************************************/

#include<stdio.h>
#include<sys/stat.h>
#include<stdlib.h>
#include<string.h>
#include<bmp.h>
#include "marca.h"


/* Funcions */
int fitBmp(bmp, bmp, int *, int *);
int pesCodi(unsigned int , int *);
int votaTota(bmp, bmp, unsigned int *, unsigned int *);
int votaRetall(bmp, bmp, unsigned int *, unsigned int *);
marca extreuMarca(unsigned int *m0, unsigned int *m1, float *marge);


/*****************************************************************************
 eMarca

 Retorna el nom del comprador d'una imatge BMP previament marcada

 descripcio:

        Donada una imatge marcada i l'original, la funcio intenta extreure
        la marca. La funcio retorna el nom del comprador i un tant per cent
        que indica el grau d'ajustament entre la marca i els vots obtinguts
        en el recompte de vots dels bits de la marca. Un valor del 100% voldra
        dir que tots els vots estan d'acord amb els bits de la marca, un 0%
        voldra dir que s'ha obtingut empat de vots i que, per tant, la marca
        obtinguda no es fiable. Val a dir que si s'ha produit mes de dos
        errors en el codi de Hamming aleshores el valor de la marca retornat
        per la funcio es erroni i el marge d'ajustament perd el seu sentit.

        La funcio te en compte que la imatge marcada hagi pogut estar
        retallada i/o invertida. Si la mida de la imatge marcada marcada es
        inferior a l'original, la funcio busca el millor encaig de la primera
        dins de   la segona tot comparant tots els bits de cada pixel excepte el
        bit inferior que, logicament, haura estat marcat. Si es troba l'encaig,
        la funcio fa el recompte de vots ja que "sap" quin es el bit de la
        marca corresponent a cada pixel.

        La funcio no contempla la possibilitat que la imatge marcada hagi
        estat sotmesa a    alteracions addicionals (addicio de so, compressio,
        etc).


 entrada:

        imgM:   punter a una instancia de tipus BMP corresponent a la imatge
                marcada
        imgS:   punter a una instancia de tipus BMP corresponent a la imatge
                original
```

```
 sortida:

          marge:   Grau d'ajustament de la marca obtinguda (0: cap; 100: perfecte)

 retorna:

          el nom del client o
          NULL si aquest no s'ha pogut trobar o be el marge de fiabilitat es
          inferior al marge minim preestablert.
*************************************************************************/

char *eMarca(bmp imgM, bmp imgS, float *marge)
{
          octet *bS;                /* On comencen les dades del BMP original */
          octet *bM;                /* On comencen les dades del BMP marcat */
          unsigned int wM;          /* Amplada (altura) del BMP marcat en pixels */
          unsigned int wS;          /* Amplada (altura) del BMP original en pixels */
          marca m;                  /* Marca */
          char *client;

          /* Vector de vots pels zeros */
          unsigned int m0[MIDA_MARCA];

          /* Vector de vots pels uns */
          unsigned int m1[MIDA_MARCA];


          /*******************************
           Parametres de les imatges
           *******************************/

          /* Buffer del BMP de la imatge original */
          bS=imgS.buf;

          /* Buffer del BMP de la imatge marcada */
          bM=imgM.buf;

          /* Amplada (altura) de la imatge marcada en pixels */
          wM=getValor(BIWIDTH_S, bM+BIWIDTH);

          /* Amplada (altura) de la imatge original en pixels */
          wS=getValor(BIWIDTH_S, bS+BIWIDTH);


          /*******************************
           Proces de votacio
           *******************************/

          /* Si la imatge marcada es mes petita que l'original pot haver
             estat retallada */
          if(wM<wS)  {
                    votaRetall(imgM, imgS, m0, m1);

                              /* Extreiem la marca */
                              m=extreuMarca(m0,m1,marge);

                              /* Cerquem a la base de dades el client. Si el client no hi es a la
                                 base de dades o el marge es insuficient, comprovem si la imatge ha
                                 estat invertida */
```

```
                        if(((client=getClient(ARXIU_DADES,
m))==NULL)||(*marge<MIN_MARGE))  {

                                /* Provem una simple reflexio horitzontal */
                                invhBmp(&imgM);
                                votaRetall(imgM, imgS, m0, m1);
                                m=extreuMarca(m0,m1,marge);

                                /* Provem una dobla reflexio horitzontal/vertical */
                                if(*marge<MIN_MARGE) {
                                        invvBmp(&imgM);
                                        votaRetall(imgM, imgS, m0, m1);
                                        m=extreuMarca(m0,m1,marge);
                                }

                                /* Provem una simple reflexio vertical */
                                if(*marge<MIN_MARGE) {
                                        invhBmp(&imgM);
                                        votaRetall(imgM, imgS, m0, m1);
                                        m=extreuMarca(m0,m1,marge);
                                }

        }


        /* La imatge no ha estat retallada */
        }  else {
                votaTota(imgM, imgS, m0, m1);

                /* Extreiem la marca */
                        m=extreuMarca(m0,m1,marge);

                        /* Cerquem a la base de dades el client. Si el client no hi es a la
                           base de dades o el marge es insuficient, comprovem si la imatge ha
                           estat invertida */
                        if(((client=getClient(ARXIU_DADES,
m))==NULL)||(*marge<MIN_MARGE))  {

                                /* Provem una simple reflexio horitzontal */
                                invhBmp(&imgM);
                                votaTota(imgM, imgS, m0, m1);
                                m=extreuMarca(m0,m1,marge);

                                /* Provem una dobla reflexio horitzontal/vertical */
                                if(*marge<MIN_MARGE) {
                                        invvBmp(&imgM);
                                        votaTota(imgM, imgS, m0, m1);
                                        m=extreuMarca(m0,m1,marge);
                                }

                                /* Provem una simple reflexio vertical */
                                if(*marge<MIN_MARGE) {
                                        invhBmp(&imgM);
                                        votaTota(imgM, imgS, m0, m1);
                                        m=extreuMarca(m0,m1,marge);
                                }
                        }

        }
```

```
	/* Amb la marca obtinguda, cerquem el client a la base de dades */
	client=getClient(ARXIU_DADES, m);

	if((client==NULL)||(*marge<MIN_MARGE)) {
		return NULL;
	} else {
		return client;
	}


}


/*****************************************************************************
 extreuMarca

 Retorna la marca d'una imatge BMP previament marcada a partir dels vectors
 de vots

 descripcio:

	A partir dels dos vectors de vots obtinguts, s'obte un marca que pot
	presentar errors en algun(s) dels seus bits. La marca aixi obtinguda
	es sotmet a una correccio, si s'escau, utilitzant els codis correctors
	de Hamming. En cas d'error doble, la marca retornada es la que es
	troba a menor distancia de les votacions obtingudes.

	La funcio retorna tambe, coma  aparametre, un tant per cent
	que indica el grau d'ajustament entre la marca i els vots obtinguts
	en el recompte de vots dels bits de la marca. Un valor del 100% voldra
	dir que tots els vots estan d'acord amb els bits de la marca, un 0%
	voldra dir que s'ha obtingut empat de vots i que, per tant, la marca
	obtinguda no es fiable. Val a dir que si s'ha produit mes de dos
	errors en el codi de Hamming aleshores el valor de la marca retornat
	per la funcio es erroni i el marge d'ajustament perd el seu sentit.


 entrada:

	m0:	vector de vots de bits igual a 0
	m1:	vector de vots de bits igual a 1

 sortida:

	marge:	Grau d'ajustament de la marca obtinguda (0: cap; 100: perfecte)

 retorna:

	el nom del client, o
	NULL si aquest no s'ha pogut trobar, o el marge de fiabilitat es
	inferior al marge minim preestablert.
*****************************************************************************/

marca extreuMarca(unsigned int *m0, unsigned int *m1, float *marge)
{

	unsigned int c[3];/* Vector de codis de Hamming corretgits */
	int pes[3];		/* Vector de pesos de cada codi */
	int pesMax;		/* Pes maxim obtingut */
	int idxPesMax;		/* Index del codi amb major pes */
```

```
int i;                          /* Comptador */
marca m;                        /* Marca */

/* Vector de diferencies de vots */
int d[MIDA_MARCA];

/*******************************
 Montem la marca
 *******************************/

/* Vector de diferencies de vots */
pesMax=0;
for(i=0;i<MIDA_MARCA;i++)  {
        pesMax+=m1[i]+m0[i];
        d[i]=m1[i]-m0[i];
}

/* Codi per DEBUG !! */
/*
printf("V0: ");
for(i=MIDA_MARCA-1;i>=0;i--)  {
        printf("%8d ", m0[i]);
}
printf("\nV1: ");
for(i=MIDA_MARCA-1;i>=0;i--)  {
        printf("%8d ", m1[i]);
}
printf("\nD: ");
for(i=MIDA_MARCA-1;i>=0;i--)  {
        printf("%8d ", d[i]);
}
printf("\n");
*/


/* Inicialitzem la marca a zero */
m=0;

/* Recorrem el vector de vots */
for(i=MIDA_MARCA-1;i>=0;i--)  {

        /* Correm els bits de la marca una posicio a l'esquerra */
        m=m<<1;

        /* Si els uns guanyen als zeros marquem el bit */
        if(m1[i]>m0[i])  {
                m=m|1;
        }

}

/*******************************
 Corretgim la marca
 *******************************/

switch(hamMarca(m,c))
{
        /* Codi correcte */
        case 0:
                /* Retornem simplement el marge de fiabilitat */
```

```
                    *marge=(float) pesCodi(m,d)*100/pesMax;
                        break;

                /* Correccio simple */
                case 1:
                        /* El codi corretgit es el primer element del
                           vector */
                        m=c[0];

                        /* Retornem el marge fiabilitat */
                        *marge=(float) pesCodi(m,d)*100/pesMax;
                        break;

                /* Correccio doble */
                case 2:
                        /* Calculem el pes de cada codi */
                        for(i=0;i<3;i++)  {
                                pes[i]=pesCodi(c[i],d);
                        }

                        /* Busquem quin es el codi amb major pes */
                        idxPesMax=0;
                        for(i=1;i<3;i++)  {
                                if(pes[i]>pes[idxPesMax]) idxPesMax=i;
                        }

                        /* Aquest es el codi amb mes pes */
                        m=c[idxPesMax];

                        /* Aquest es el seu marge de fiabilitat */
                        *marge=(float) pes[idxPesMax]*100/pesMax;
                        break;

                default:
                        break;

        }

        /* Retornem la marca */
        return m;

}
```

```
/*****************************************************************************
 fitBmp

        Calcula el millor encaix d'una porcio d'una imatge retallada dins
        de la imatge original.

 descripcio:

        Donada una imatge, i un retall de la mateixa, volem saber les
        coordenades corresponents al tall. Per aixo, comparem octet a octet
        ignorant el bit de menys pes ja que aquest es un bit de marca. Si
        despres de comparar els 7 bits de mes pes de cada entre cada octet
        de la imatge retallada i els octets que formen un quadrat de la imatge
        original que comencen en el pixel (u,v), veiem que no hi ha cap
        diferencia, podrem deduir que el pixel (u,v) ens defineix les
        coordenades de la porcio dins de l'original.

        La comparacio nomes es perfecte si la imatge retallada no ha estat
        sotmesa a retocs addicionals.

 entrada:

        imgM:   instancia del BMP marcat
        imgS:   instancia del BMP original

 sortida:

        u,v:    coordenades del primer pixel de l'encaix

 retorna:

        0 si s'ha trobat un encaix
        -1 si no s'ha trobat cap encaix
*****************************************************************************/

int fitBmp(bmp imgM, bmp imgS, int *u, int *v)
{

        octet *bS;              /* On comencen les dades del BMP original */
        octet *bM;              /* On comencen les dades del BMP marcat */
        unsigned int wM;        /* Amplada (altura) del BMP marcat en pixels */
        unsigned int wS;        /* Amplada (altura) del BMP original en pixels */
        int x,y;                /* Coordenades d'un pixel de la imatge original */
        int ix,jx;              /* Coordenades d'un pixel de la imatge marcada */
        unsigned int oct;       /* Index per un octet de la imatge marcada */
        unsigned int octM,octS; /* Octets a comparar */
        int fet;                /* Boolea per saber si s'ha acabat una
                                   comparacio d'encaix */

        /* Buffer del BMP de la imatge marcada */
        bM=imgM.buf;

        /* Buffer del BMP de la imatge original */
        bS=imgS.buf;

        /* Amplada (altura) de la imatge marcada en pixels */
        wM=getValor(BIWIDTH_S, bM+BIWIDTH);
```

```
/* Amplada (altura) de la imatge original en pixels */
wS=getValor(BIWIDTH_S, bS+BIWIDTH);

/* On comencen les dades de la imatge original */
bS+=getValor(BFOFFBITS_S, bS+BFOFFBITS);

/* On comencen les dades de la imatge marcada */
bM+=getValor(BFOFFBITS_S, bM+BFOFFBITS);

fet=0;

/* Comencem pel primer pixel de la imatge original */
x=0;
y=0;

/* Aquest bucle s'acaba quan hem trobat un encaix o hem esgotat
   els pixels */
while(-1)  {

        /* Comencem pel primer pixel de la imatge marcada */
        ix=0;
        jx=0;
        oct=0;

        /* Mentre no esgotem els pixels i no haguem trobat
           un encaix... */
        while((ix<wM)&&(jx<wM)&&(!fet))  {

                /* Comparem els 7 bits de mes pes de l'octet R */
                octM=bM[oct++]&0xfe;
                octS=bS[((x+ix)*wS+y+jx)*3]&0xfe;
                if(octM!=octS)  break;

                /* Comparem els 7 bits de mes pes de l'octet G */
                octM=bM[oct++]&0xfe;
                octS=bS[((x+ix)*wS+y+jx)*3+1]&0xfe;
                if(octM!=octS)  break;

                /* Comparem els 7 bits de mes pes de l'octet B */
                octM=bM[oct++]&0xfe;
                octS=bS[((x+ix)*wS+y+jx)*3+2]&0xfe;
                if(octM!=octS)  break;

                /* Pixel marcat seguent */
                jx++;

                /* Cal canviar de fila? */
                if(jx==wM)   {
                        ix++;

                        /* Si arrivem aqui hem trobat l'encaix */
                        if(ix==wM) {
                                *u=x;
                                *v=y;
                                fet=-1;
                                return 0;
                        } else {
                                jx=0;
                        }
```

```
                        }
                }

                /* Pixel original seguent */
                y++;

                /* Cal passar a la fila seguent? */
                if(y>wS-wM)  {
                        x++;

                        /* Si arribem al final, no hem trobat cap encaix */
                        if(x>wS-wM)  return -1;

                        /* Comencem una nova fila */
                        y=0;
                }

        }

}


/*****************************************************************************
 pesCodi

        Calcula el pes d'un codi de Hamming en relacio al vector de
        diferencies de vots

 descripcio:

        Donat un codi de Hamming dual, volem saber en quin grau s'ajusta
        als vots obtinguts en el proces d'extraccio de la marca. Per aixo
        es crea un vector de diferencies, cada component de les quals es
        la diferencia de les components corresponents dels vectors m1 i m0.
        Un valor positiu d'un element d'aquest vector de diferencies indicara
        que el nombre de vots corresponents al bit 1 es superior als del
        bit 0, i a l'inreves. El pes del codi es un enter igual a la suma de
        diferencies en valor absolut cadascuna ponderada per un signe + si
        el bit del codi esta en acord amb la diferencia i per un signe - si
        esta en desacord amb la diferencia corresponent.

        Exemple:

        Suposem que els vectors de vots han estat els seguents:

        m0:     10      5       50      120     1000    90      40
        m1:      2      60      52      140     500     81      42

        El vector de diferencies seria:

        d:      -8      55      2       20      -500    -9      2

        Suposem ara que volem saber en quin grau s'ajusta el codi

                1110010

        als resultats anteriors. Calculem el pes de la manera seguent:

                pes(1110010) = -8 +55 +2 -20 +500 -9 -2 = 518
```

```
        Es a dir, els valors absoluts de les diferencies se sumen si el
        bit del codi coincideix amb el bit amb mes vots i es resta en cas
        contrari. (Una diferencia de vots igual a zero li assignarem el
        bit 1!).

 entrada:

        c:      codi de Hamming dual
        d:      vector de diferencies de vots

 retorna:

        pes del codi
*****************************************************************************/

int pesCodi(unsigned int c, int *d)
{

        int pes;  /* Pes del codi */
        int i;            /* Comptador */

        /* Posem el pes a zero */
        pes=0;

        /* Per a cada bit del codi de Hamming dual... */
        for(i=0;i<MIDA_MARCA;i++)  {

                /* Si el bit de menys pes es 0... */
                if(c%2==0)  {

                        /* i la diferencia de vots corresponent a a aquest bit
                           es negativa (es a dir, guanyen els 0's... */
                        if(d[i]<0)  {

                                /* el pes augmenta en el valor absolut
                                   d'aquest diferencia */
                                pes+=abs(d[i]);

                        /* si la diferencia es positiva, hem de restar-li */
                        }  else  {
                                pes+=-d[i];
                        }

                /* Si el bit de menys pes es 1... */
                }  else  {
                        /* Tant si la diferencia corresponent es positiva o
                           negativa, hem de sumar-la al pes */
                        pes+=d[i];
                }

                /* Passem al bit seguent del codi */
                c=c>>1;
        }

        /* Retornem el pes */
        return pes;
}
```

```
/*****************************************************************************
 votaTota

 Recompta els vots de la marca per a una imatge marcada sencera

 entrada:

          imgM:   punter a una instancia de tipus BMP corresponent a la imatge
                  marcada
          imgS:   punter a una instancia de tipus BMP corresponent a la imatge
                  original

 sortida:

          m0:     vector de vots de bits igual a 0
          m1:     vector de vots de bits igual a 1



 retorna:

          0
*****************************************************************************/

int votaTota(bmp imgM, bmp imgS, unsigned int *m0, unsigned int *m1)
{

          octet *bS;              /* On comencen les dades del BMP original */
          octet *bM;              /* On comencen les dades del BMP marcat */
          unsigned int wM;        /* Amplada (altura) del BMP marcat en pixels */
          int pixQ;               /* Pixels per costat de cada quadrat del
                                     BMP marcat */
          int pixC;               /* Pixels per costat de cada quadret del
                                     BMP marcat */
          int x,y;                /* Coordenades en pixels d'un quadrat del
                                     BMP marcat */
          int fil, col;           /* Copies de x i y */

          int qi,qj;              /* Quadrat generic ij */
          int ci,cj;              /* Quadret generic ij */
          int iByt,jByt;          /* Octet ij dins d'un quadret */
          int oct;                /* Index per a un octet del BMP marcat */
          octet bitM;             /* Bit de menys pes d'un octet del BMP marcat */
          octet bitS;             /* Bit de menys pes d'un octet del BMP original */
          int bitsCanviats;       /* Per a cada bit de la marca, nombre de bits
                                  diferents entre la imatge marcada i
                                  l'original */
          int iBit;               /* Index pels vectors de vots */
          int i;                  /* Comptador */


          /*********************************
           Proces de recompte de vots
           *********************************/

          /* Inicialitzem els vectors de vots a zero */
          for(i=0;i<MIDA_MARCA;i++)  {
                    m0[i]=0;
                    m1[i]=0;
```

```
}

/* Buffer del BMP de la imatge original */
bS=imgS.buf;

/* Buffer del BMP de la imatge marcada */
bM=imgM.buf;

/* Amplada (altura) de la imatge marcada en pixels */
wM=getValor(BIWIDTH_S, bM+BIWIDTH);

/* On comencen les dades de la imatge original */
bS+=getValor(BFOFFBITS_S, bS+BFOFFBITS);

/* On comencen les dades de la imatge marcada */
bM+=getValor(BFOFFBITS_S, bM+BFOFFBITS);

/* Subividim cada costat de la imatge marcada en 4 quadrats */
pixQ=wM>>2;

/* Subividim cada quadrat de la imatge marcada en 100 quadrets */
pixC=pixQ/10;

/* Per a cada filera de quadrats... */
for(qi=0;qi<4;qi++)   {

  /* Per a cada columna de quadrats... */
  for(qj=0;qj<4;qj++)   {

        /* Comencem a llegir el primer bit de la marca */
        iBit=0;

        /* Per a cada filera de quadrets... */
        for(ci=0;ci<10;ci++)   {

   /* Cada fila del BMP marcat comenca en aquest pixel */
   x=qi*pixQ+ci*pixC;

      /* Guardem aquesta fila */
      fil=x;

      /* Per a cada columna de quadrets... */
          for(cj=0;cj<10;cj++)   {

                /* Cada columna del BMP marcat comenca en aquest pixel */
                y=qj*pixQ+cj*pixC;

                /* Guardem aquesta columna */
                col=y;

                /* Per a cada fila de pixels del quadret...*/
                for(iByt=0;iByt<pixC;iByt++)  {

                  /* Per a cada columna de pixels del quadret...*/
                  for(jByt=0;jByt<pixC;jByt++)  {

                        /* Comptem quants bits van canviant */
                        bitsCanviats=0;

                        oct=3*(x*wM+y);
```

```
                        /* Ha canviat l'octet R? */
                        bitM=bM[oct]&1;
                        bitS=bS[oct]&1;
                        if(bitM!=bitS)  {
                                bitsCanviats++;
                        }

                        /* Ha canviat l'octet G? */
                        bitM=bM[oct+1]&1;
                        bitS=bS[oct+1]&1;
                        if(bitM!=bitS)  {
                                bitsCanviats++;
                        }

                        /* Ha canviat l'octet B? */
                        bitM=bM[oct+2]&1;
                        bitS=bS[oct+2]&1;
                        if(bitM!=bitS)  {
                                bitsCanviats++;
                        }

                        if(bitsCanviats>1)  {
                                m1[iBit]++;
                        } else  {
                                m0[iBit]++;
                        }

                        y++;
                    }

                    /* Seguent fila de pixels */
                    x++;

                    /* Restituim la columna */
                    y=col;
                }

                /* Restituim la fila */
                x=fil;

                /* Seguent bit de la marca */
                if(iBit==MIDA_MARCA-1)  {
                        iBit=0;
                } else  {
                        iBit++;
                }

            }
        }
    }
}

/* Fet */
return 0;

}
```

```
/*************************************************************************
 votaRetall

 Recompta els vots de la marca per a una imatge marcada retallada

 entrada:

          imgM:   punter a una instancia de tipus BMP corresponent a la imatge
                  marcada
          imgS:   punter a una instancia de tipus BMP corresponent a la imatge
                  original

 sortida:

          m0:     vector de vots de bits igual a 0
          m1:     vector de vots de bits igual a 1

 retorna:

          0
*************************************************************************/

int votaRetall(bmp imgM, bmp imgS, unsigned int *m0, unsigned int *m1)
{
          octet *bS;                /* On comencen les dades del BMP original */
          octet *bM;                /* On comencen les dades del BMP marcat */
          unsigned int wM ;         /* Amplada (altura) del BMP marcat en pixels */
          unsigned int wS;          /* Amplada (altura) del BMP original en pixels */
          int pixQ;                 /* Pixels per costat de cada quadrat del
                                      BMP marcat */
          int u,v;                  /* Per a imatges marcades retallades, les
                                     coordenades la porcio respecte de la imatge
                                    original */
          int x,y;                  /* Posicio d'un pixel de la imatge original */
          int ix,jx;                /* Posicio d'un pixel de la imatge marcada */
          int qx,qy;                /* Quadre al que pertany un pixel de la imatge
                                    marcada */
          int cx,cy;                /* Quadret al que pertany un pixel de la
                                      imatge marcada */
          unsigned int octM;        /* Index pels octets de la imatge marcada */
          unsigned int octS;        /* Index pels octets de la imatge original */
          unsigned int bitM;        /* Bit inferior d'un octet de la imatge marcada */
          unsigned int bitS;        /* Bit inferior d'un octet de la imatge original */
          int iBit;                 /* Index pels vectors de vots */
          int bitsCanviats;         /* Per a cada bit de la marca, nombre de bits
                                    diferents entre la imatge marcada i
                                    l'original */
          int i;                    /* Comptador */


          /*********************************
           Parametres de les imatges
           *********************************/

          /* Buffer del BMP de la imatge marcada */
          bM=imgM.buf;
```

```
/* Buffer del BMP de la imatge original */
bS=imgS.buf;

/* Amplada (altura) de la imatge marcada en pixels */
wM=getValor(BIWIDTH_S, bM+BIWIDTH);

/* Amplada (altura) de la imatge original en pixels */
wS=getValor(BIWIDTH_S, bS+BIWIDTH);

/* On comencen les dades de la imatge original */
bS+=getValor(BFOFFBITS_S, bS+BFOFFBITS);

/* On comencen les dades de la imatge marcada */
bM+=getValor(BFOFFBITS_S, bM+BFOFFBITS);

/* Subividim cada costat de la imatge original en 4 quadrats */
pixQ=wS>>2;


if(fitBmp(imgM,imgS,&u,&v)!=0)  {
                    u=0;
                    v=0;
}


/*********************************
 Votacions de bits de la marca
 *********************************/

/* Inicialitzem els vectors de vots a zero */
for(i=0;i<MIDA_MARCA;i++)  {
          m0[i]=0;
          m1[i]=0;
}


/* Comencem pel primer pixel de la imatge marcada */
ix=0;
jx=0;
octM=0;

/* Mentre hi ha hagi pixels per comparar... */
/*while(ix*jx<(wM-1)*(wM-1))  {*/
while(octM<wM*3*wM-1)  {

          /* Posicio del pixel de la imatge original */
          x=u+ix;
          y=v+jx;

          /* Quadre a que pertany aquest pixel */
          qx=x%pixQ;
          qy=y%pixQ;

          /* Quadret a que pertany el pixel */
          cx=qx/10;
          cy=qy/10;
```

```
            /* Bit de la marca que licorrespon a aquest pixel */
            iBit=(cx*10+cy)%MIDA_MARCA;

            /* Comptem quants bits van canviant */
            bitsCanviats=0;

            /* Octet de la imatge original a comparar */
            octS=(x*wS+y)*3;

            /* Ha canviat l'octet R? */
            bitM=bM[octM++]&1;
            bitS=bS[octS++]&1;
            if(bitM!=bitS) bitsCanviats++;

            /* Ha canviat l'octet G? */
            bitM=bM[octM++]&1;
            bitS=bS[octS++]&1;
            if(bitM!=bitS) bitsCanviats++;

            /* Ha canviat l'octet B? */
            bitM=bM[octM++]&1;
            bitS=bS[octS++]&1;
            if(bitM!=bitS) bitsCanviats++;

            /* Si hi ha hagut mes d'un canvi, hi ha un bit de marca */
            if(bitsCanviats>1)  {
                        m1[iBit]++;
            } else  {
                        m0[iBit]++;
            }

            /* Passem a la columna seguent */
            jx++;

            /* Si hem arribat al final de la fila, passem a la seguent */
            if(jx==wM)   {
                        ix++;
                        jx=0;
            }

      }

      /* Fet */
      return 0;


}
```

```
/*****************************************************************************
 hamMarca.c

 TREBALL FI DE CARRERA - PROTECCIO DEL COPYRIGHT

 Jordi Trubat Dalmau, jtrubat@uoc.edu
*****************************************************************************/

#include<stdio.h>
#include<string.h>

/* Funcio paritat */
int paritat(unsigned int);

/*****************************************************************************
 hamMarca

 descripcio:

        La funcio comprova si el codi dual de Hamming de 7 bits passat com a
        argument, tealgun error. La funcio retorna el valor 0 si no hi ha
        error, 1 si hi ha error simple, 2 si hi ha error doble.
        La funcio retorna com a parametre(s) el(s) error(s) corretgit(s) en
        el cas que n'hi hagi hagut.

 observacions:

        Per codificar una paraula de 3 bits de manera que ens permeti
        corretgir errors simples i detectar errors dobles, afegim als 3 bits
        d'informacio 4 bits de paritat:

                x x x c1 c2 c3 p

        Si enumerem els bits d'esquerra a dreta i de 1 a 7, tenim que

        c1: paritat dels bits 1, 2 i 3
        c2: paritat dels bits 1 i 2
        c3: paritat dels bits 1 i 3
        p:  paritat dels bits 1 a 6

        P. ex, si volem codificar 110 tindrem que c1=0, c2=0, c3=1, p=0:

                110 001 1

        Per detectar errors, construim un numero binari de 4 bits anomenat
        sindrome que conte els valors de paritat seguents:

                s1 s2 s3 s4

        s1: paritat dels bits 1, 2, 3 i 4 del codi
        s2: paritat dels bits 1, 2 i 5 del codi
        s3: paritat dels bits 1, 3 i 6 del codi
        s4: paritat de tots els bits del codi

        Es evident que si tots aquests bits son zero el codi es correcte. Si
        hi ha hagut algun error (simple o doble) el sindrome sera diferent
        de zero. P. ex. un sidrome igual a 0111 voldra dir que hi ha hagut
        un error simple en el segon bit del codi; un sindrome igual a 1100
        voldra dir que hi ha hagut error doble: o be en els bits 3 i 5, o be
```

en els bits 2 i 6, o be en els bits 1 i 7.

En general, per a codis de Hamming duals de 7 bits, tindrem que si l'error es simple, podrem corregir l'error. Si l'error es doble, podrem donar 3 correccions possibles. Observem que si l'error es simple, el bit de menys pes del sindrome (bit s4) sempre sera 1, ja que la paritat del codi sera incorrecta. Si l'error es doble aquest bit sempre sera 0 (dos errors donen lloc a una paritat del codi 'correcta'). Aquest fet ens permet utilitzar el sindrome per indexar dues taules de correccio d'errors, una d'errors simples i una altra d'errors dobles. Aquestes taules contenen entrades que OR'ejades' exclusivament amb el codi erroni ens corretgeix l'error.

Aixi un cop hem calculat el sidrome i hem vist que no es zero, mirem quin es el seu bit de menys pes. Eliminem aquest bit del sindrome i el valor que resta fa d'index per a la taula corresponent. La taula d'errors simples tenen una entrada per a cada valor de l'index, i per tant la correccio es unica. La taula d'errors dobles es en realitat una matriu de tantes files com valors pot prendre l'index, i tres columnes, una per a cada codi de correccio possible. (Observem que a l'entrada 111 en binari de la taula de correccio d'errors simple hi hem posat un zero, ja que el sindrome 1110 es l'unic que no dona lloc a cap error simple!).

Exemple:

- Error simple

Suposem que el codi de Hamming dual de l'exemple d'abans

1100011

presenta un error simple en el primer bit:

0100011

El sindrome valdra:

1101

ja que:

s1 (paritat dels bits 1, 2, 3 i 4) = 1
s2 (paritat dels bits 1, 2 i 5 ) = 1
s3 (paritat dels bits 1, 3 i 6) = 1
s4 (paritat de tots els bits) = 1

que s4 sigui 1 ens diu que l'error es simple. Aleshores eliminant aquest bit del sindrome i utilitzant el valor resultant

111 = 7 (en decimal)

com a index de la taula de correccio d'errors simples, obtenim l'entrada:

errSimple[7] = 64

que en binari es

errSimple[7] = 1000000

Si ara fem

codi XOR errSimple[5] = 0100011 ^ 1000000 = 1100011

Observem que recuperem el codi original.

- Error doble

Si l'error es doble, per exemple:

0100010

(error en els bits 1 i 7), el sindrome sera

1100

ja que:

s1 (paritat dels bits 1, 2, 3 i 4) = 1
s2 (paritat dels bits 1, 2 i 5 ) = 1
s3 (paritat dels bits 1, 3 i 6) = 1
s4 (paritat de tots els bits) = 0

Com que s4 es 0, l'error es doble . Aleshores eliminant aquest bit del sindrome i utilitzant el valor resultant

111 = 7 (en decimal)

menys 1 (el valor 000 correspondria a un codi correcte, i per tant, no s'ha afegit a la taula) com a index de fila de la matriu de correccio d'errors dobles, obtenim les tres entrades corresponents:

errDoble[6,0] = 20
errDoble[6,1] = 34
errDoble[6,2] = 65

que en binari es

errDoble[6,0] = 0010100
errDoble[6,1] = 0100010
errDoble[6,2] = 1000001

Si ara fem

codi XOR errDoble[6,0] = 0100010 ^ 0010100 = 0110110
codi XOR errDoble[6,1] = 0100010 ^ 0100010 = 0000000
codi XOR errDoble[6,2] = 0100010 ^ 1000001 = 1100011

Observem que nomes la tercera correccio es la bona.

entrada:

codi: codi de Hamming

sortida:

c: vector de codis de Hamming corretgits

```
 retorna:

          0 si el codi passa les comprovacions de paritat
          1 si el codi te un error simple
          2 si el codi te un error doble
*****************************************************************************/

int hamMarca(unsigned int codi, unsigned int *c)
{

          unsigned int synd;          /* Sindrome del codi */
          unsigned int t;             /* Temporal per al codi de Hamming */

          /* Taula de codis correctors d'errors simples */
          unsigned int errSimple[]={1,8,4,32,2,16,0,64};

          /* Taula de codis correctors d'errors dobles */
          unsigned int errDoble[]={18,36,9,
                                   40,80,5,
                                   33,66,12,
                                   24,96,3,
                                   17,68,10,
                                   72,48,6,
                                   20,34,65};


          /****************************************
           Montem el sidrome
           ****************************************/

          /* Inicialitzem el sindrome */
          synd=0;

          /* Paritat de tots els bits */
          t=codi;
          t=t&0xffff;
          if(paritat(t)!=0)  synd=1;

          /* Paritat dels bits 1, 2, 3 i 4 */
          t=codi;
          t=t&120;
          if(paritat(t)!=0)  synd=synd|2;

          /* Paritat dels bits 1, 2 i 5 */
          t=codi;
          t=t&100;
          if(paritat(t)!=0)  synd=synd|4;

          /* Paritat dels bits 1, 3 i 6 */
          t=codi;

          t=t&82;
          if(paritat(t)!=0)  synd=synd|8;
```

```
        /****************************************
         Codi correcte?
         ****************************************/

        /* Si el sindrome es zero, codi correcte */
        if(synd==0)  {
                return 0;
        }

        /****************************************
         Correccio del codi
         ****************************************/

        /* Si el sindrome es senar, error simple */
        if(synd%2!=0)   {

                /* Corretgim l'error simple */
                c[0]=codi^errSimple[synd>>1];
                return 1;

        /* Si el sindrome es parell, error doble */
        } else {

                /* Hi ha tres possibles correccions */
                c[0]=codi^errDoble[((synd>>1)-1)*3];
                c[1]=codi^errDoble[((synd>>1)-1)*3+1];
                c[2]=codi^errDoble[((synd>>1)-1)*3+2];
                return 2;
        }

}


/*****************************************************************************
 paritat

 Calcula la paritat d'un codi binari

 descripcio:

        La paritat d'un codi binari es zero si el nombre de bits actius es
        parell, 1 si es senar. Per a comptar els bits del codi en llenguatge
        C fem servir la tecnica seguent (PC Assembly Language, Paul Carter,
        pag. 53). Si el codi te el format

                codi =   xxxxx10000

        on x es 0 o 1, codi-1 sera:

                codi-1 = xxxxx01111

        Si a continuacio fem l'operacio logica

                codi & (codi-1)   = xxxxx00000

        haurem eliminat el bit actiu de menys pes. Si mantenim un comptador,
        podrem saber el nombre de bits actius que te el codi fins que aquest
        esdeve igual a zero.
```

```
 entrada:

            codi:      codi binari

 retorna:
            0          si la paritat es parella
            1          si la paritat es senar
*****************************************************************************/

int paritat(unsigned int codi)  {

            int bitsOn=0;                    /* Nombre de bits actius del codi */

            /* Mentre quedin bits actius... */
            while(codi!=0)  {

                        /* Eliminem el bit de menys pes */
                        codi=codi&(codi-1);

                        /* Actualitzem el comptador */
                        bitsOn++;
            }

            /* Si el nombre bits es parell, paritat 0 */
            if(bitsOn%2==0)  {
                        return 0;

            /* Si el nombre bits es senar, paritat 1 */
            }  else  {
                        return 1;
            }

}
```

```
/*****************************************************************************
 bdMarca.c

 TREBALL FI DE CARRERA - PROTECCIO DEL COPYRIGHT

 Jordi Trubat Dalmau, jtrubat@uoc.edu
*****************************************************************************/

#include<stdio.h>
#include<sys/stat.h>
#include<stdlib.h>
#include<string.h>
#include<bmp.h>
#include "marca.h"


/*****************************************************************************
 getMarca

 Cerca la marca (tira de bits) del client passat com a parametre de la
 base de dades passada tambe com a parametre

  descripcio:

          Quan volem encastar una imatge amb una marca a partir del nom del
          comprador, hem de consultar la bases de dades per saber quina es
          la marca que correspon al comprador. Si la cerca reixeix, la rutina
          retorna la marca associada al comprador. Si el comprador no hi es a
          la base de dades, ess retorna un valor nul.

 entrada:

          fitxer:    nom de la base de dades
          client: nom del client

 sortida:

          m:         marca

 retorna:

          0 si existeix el client i s'ha pogut retornar la marca
          -1 en cas d'error
*****************************************************************************/

int getMarca(char *fitxer, char *client, marca *m)
{

          FILE *f;                  /* Descriptor del fitxer */
          int retv;                 /* Valor retornat per fscanf */
          marca tMarca;             /* Temporal per una marca */
          char tClient[126];        /* Temporal per a un client */





          /* Obrim la base de dades */
```

```
	if((f=fopen(fitxer,"r"))==NULL)	{
		errMarca(E_MAR_FTXINOBRIR);
		return -1;
	}

	/* Llegim els registres de la base de dades */
	while(((retv=fscanf(f,"%s %d",tClient,&tMarca))!=EOF)&&(strcmp(client,tClient)!=0));

	/* Tanquem el fitxer */
	fclose(f);

	/* Si hem arribat a la marca EOF, el client no existeix */
	if(retv==EOF)	{
		errMarca(E_MAR_CLIENTINC);
		return -1;
	}

	/* Retornem la marca */
	*m=tMarca;

	/* Fet */
	return 0;

}



/*****************************************************************************
 getClient

 Cerca el client corresponent a una marca passada com a parametre de la
 base de dades passada tambe com a parametre

 descripcio:

	Una vegada s'ha llegit una marca d'una imatge mitjancant la rutina
	eMarca(), interessara saber el nom del comprador. Per aixo cal
	consultar la base de dades i veure quin es el nom que va lligat a
	la marca. Si la cerca reixeix, la rutina retorna el nom del
	comprador. En cas contrari retorna un punter nul.

 entrada:

	fitxer:	nom de la base de dades
	m:	marca

 retorna:

	client:	punter al client
	NULL:	la marca no correspon a cap client
*****************************************************************************/

char *getClient(char *fitxer, marca m)
{

	FILE *f;		/* Descriptor del fitxer */
	int retv;		/* Valor retornat per fscanf */
	marca tMarca;		/* Temporal per una marca */
	char tClient[125];	/* Temporal per a un client */
	char *client;		/* Punter al client trobat */
```

```
	/* Obrim la base de dades */
	if((f=fopen(fitxer,"r"))==NULL)  {
		errMarca(E_MAR_FTXINOBRIR);
		return NULL;
	}

	/* Llegim els registres de la base de dades */
	while(((retv=fscanf(f,"%s %d",tClient, &tMarca))!=EOF)&&(m!=tMarca));

	/* Tanquem el fitxer */
	fclose(f);

	/* Si hem arribat a la marca EOF, la marca no hi es */
	if(retv==EOF)  {
		return NULL;
	}

	/* Creem espai per guardar el client */
	if((client=(char*)malloc(strlen(tClient)))==NULL)  {
		errMarca(E_MAR_NOMEM);
		return NULL;
	}



	/* Copiem el client a l'espai de memoria creat */
	client=strncpy(client, tClient, strlen(tClient));

	/* Retornem el punter al client */
	return client;

}
```

```
/*****************************************************************************
 errMarca.c

 TREBALL FI DE CARRERA - PROTECCIO DEL COPYRIGHT

 Jordi Trubat Dalmau, jtrubat@uoc.edu
*****************************************************************************/

#include<stdio.h>
#include<bmp.h>
#include "marca.h"


/*****************************************************************************
 errMarca

 Imprimeix al canal estandard d'error el missatge d'error corresponent al
 codi passat com a parametre

 descripcio:

        L'objectiu de la funcio es mantenir informat a l'usuari (o al
        desenvolupador) de l'origen dels errors o excepcions de programari
        que puguin sorgir durant la seva execucio. La funcio imprimeix al
        canal estandard d'error un missatge explicatiu de l'error o excepcio
        produida. El missatge va lligat amb el codi d'error passat com a
        parametre. Els codis i els missatges estan definits al fitxer marca.h.

 entrada:

        codi:   codi d'error

 retorna:

        0
*****************************************************************************/

int errMarca(int codi)
{
        switch(codi)
        {
        case E_MAR_NPARAMETRESINC:
                fprintf(stderr, E_MAR_NPARAMETRESINC_M);
                break;

        case E_MAR_PARAMETREINC:
                fprintf(stderr, E_MAR_PARAMETREINC_M);
                break;

        case E_MAR_EXTINC:
                fprintf(stderr, E_MAR_EXTINC_M);
                break;

        case E_MAR_CLIENTINC:
                fprintf(stderr, E_MAR_CLIENTINC_M);
                break;

        case E_MAR_IMATGESDIF:
```

```
                fprintf(stderr, E_MAR_IMATGESDIF_M);
                break;

        case E_MAR_NOMEM:
                fprintf(stderr, E_MAR_NOMEM_M);
                break;

        case E_MAR_FTXINOBRIR:
                fprintf(stderr, E_MAR_FTXINOBRIR_M);
                break;

        case E_MAR_FTXINTANCAR:
                fprintf(stderr, E_MAR_FTXINTANCAR_M);
                break;




        case E_MAR_MASSACONFS:
                fprintf(stderr, E_MAR_MASSACONFS_M);
                break;

        /* Si no es cap dels anterios, imprimim un "error desconegut"*/
        default:
                fprintf(stderr, E_MAR_DESC_M);

        }

        return 0;
}
```

```
/*****************************************************************************
 confabula.c

 TREBALL FI DE CARRERA - PROTECCIO DEL COPYRIGHT

 Jordi Trubat Dalmau, jtrubat@uoc.edu
*****************************************************************************/

#include<stdio.h>
#include<stdlib.h>
#include<time.h>
#include<sys/stat.h>
#include<string.h>
#include<bmp.h>
#include "marca.h"


/* Funcio conf */
int conf(int,bmp *);

/*****************************************************************************
 nom:

        confabula - Confabula una imatge a partir de dues o mes imatges
                    marcades per un mateix venedor a diferents compradors

 sinopsi:

        confabula imatge1 imatge2 imatge3 ...

 descripcio:

        Es confabula una imatge a partir de dues o mes imatges BMP de
        24 bits, quadrades, sense comprimir, marcades per un mateix venedor
        i venudes a diferents compradors. La funcio compara pixel a pixel les
        imatges. Si els pixels comparats difereixen, es canvia aleatoriament
        el bit de menys pes de cada octet del pixel de la primera imatge, que
        esdevindra la imatge confabulada
*****************************************************************************/


int main(int argc, char **argv)
{

        unsigned long mida;             /* Mida d'un fitxer BMP */
        octet *buf;                     /* Punter al buffer on es guarda un BMP */
        char *imatge;                   /* Nom de la imatge a confabular */
        char *temp;                     /* Temporal de nomd de les altres imatges */
        char *imatgeConf;               /* Nom de la imatge confabulada */
        char *client[MAX_CONFS];        /* Vector de nosm dels compradors */
        bmp bmp[MAX_CONFS];             /* Vector d'instancies BMP de les imatges */
        int midaCadena;                 /* Mida de la cadena del nom de la imatge
                                           confabulada */
        int i;                          /* Comptador */

        /* Nombre de parametres correcte? */

        /* Com a minim hi ha d'haver 2 confabuladors */
        if(argc<3)  {
```

```
		errMarca(E_MAR_NPARAMETRESINC);
		return -1;
	}

	/* Pero tambe hi ha un maxim de confabuladors */
	if(argc>MAX_CONFS+1)  {
		errMarca(E_MAR_MASSACONFS);
		return -1;
	}

	/* Extensio de les imatges correcta? */
	for(i=1;i<argc;i++)  {
		if((strstr(argv[i],"bmp")==NULL)&&(strstr(argv[i],"BMP")==NULL)) {
			errMarca(E_MAR_EXTINC);
			return -1;
		}
	}

	/* Nom de la imatge del primer comprador */
	if((imatge=(char*)malloc(strcspn(argv[1],".")+1))==NULL)  {
		errMarca(E_MAR_NOMEM);
		return -1;
	}
	imatge=strncpy(imatge, argv[1], strcspn(argv[1],"."));

	/* Nom de les imatges de la resta de compradors */
	for(i=2;i<argc;i++)  {

		if((temp=(char*)malloc(strcspn(argv[i],".")+1))==NULL)  {
			errMarca(E_MAR_NOMEM);
			return -1;
		}
		temp=strncpy(temp, argv[i], strcspn(argv[i],"."));

		/* Es tracta de la mateixa imatge que la primera? */
		if(strcmp(imatge,temp)!=0)  {
			errMarca(E_MAR_IMATGESDIF);
			return -1;
		}

	}



	/****************************************
	 Vector de noms dels compradors
	 ****************************************/

	/* Nom dels clients */
	for(i=1;i<argc;i++)  {
		if((client[i-1]=(char*)malloc(strlen(argv[i])-strlen(imatge)-5+1))==NULL)  {
			errMarca(E_MAR_NOMEM);
			return -1;
		}
		client[i-1]=strncpy(client[i-1], argv[i]+strlen(imatge)+1, strlen(argv[i])-strlen(imatge)-
5);
	}
```

```
/****************************************
 Montem el nom de la imatge confabulada
 ****************************************/

/* Primer calculem la longitud de la cadena de la imatge confabulada */

/* Nom de la imatge + el . */
midaCadena=strlen(imatge)+1;

/* Etiqueta conf + el . */
midaCadena+=strlen("conf")+1;

/* Nom de cada comprador + el . */
for(i=1;i<argc;i++)  {
        midaCadena+=strlen(client[i-1])+1;
}

/* Extensio */
midaCadena+=strlen("bmp");

/* Fem lloc per aquesta cadena */
if((imatgeConf=(char*)malloc(midaCadena+1))==NULL)  {
        errMarca(E_MAR_NOMEM);
        return -1;
}

/* Montem la cadena */
imatgeConf=strcat(imatgeConf, imatge);
imatgeConf=strcat(imatgeConf, ".conf.");
for(i=1;i<argc;i++)  {
        imatgeConf=strcat(imatgeConf, client[i-1]);
        imatgeConf=strcat(imatgeConf, ".");
}
imatgeConf=strcat(imatgeConf, "bmp");


/****************************************
 Vector de BMPs
 ****************************************/
/* Carreguem les imatges a la memoria */
for(i=1;i<argc;i++)  {
        if((buf=loadBmp(&mida, argv[i]))==NULL)  {
                return -1;
        }

        /* Omplim els camps de l'estructura BMP */
        bmp[i-1].nom=argv[i];
        bmp[i-1].buf=buf;

        /* Llegim l'estructura de cada BMP */
        if(chkBmp(bmp[i-1])!=0)  {
                return -1;
        }
}
```

```
        /****************************************
         Confabulem i guardem la imatge confabulada
         ****************************************/

        /* Hem de confabular argc-1 imatges */
        conf(argc-1, bmp);

        /* El primer BMP es el confabulat. Li posem el nou nom */
        bmp[0].nom=imatgeConf;

        /* Guardem el BMP confabulat */
        if(saveBmp(bmp[0])!=0)  {
                return -1;
        }

        /* Fet */
        return 0;

}
```

```
/*************************************************************************
 conf

 Confabula una imatge a partir de dues imatges comprades a un mateix venedor

 entrada:

          img1:    primera instancia tipus BMP
          img2:   segona instancia tipus BMP

 sortida:

          img1:    instancia de la imatge confabulada

 retorna:
          0
*************************************************************************/

int conf(int n, bmp *img)  {


          octet *b[MAX_CONFS];  /* Punters als buffers de les dues imatges */
          unsigned int pix;             /* Comptador de pixels */
          unsigned int oct;             /* Index per als octets de les imatges */
          unsigned char bit;            /* Pixel aleatori */
          int pixDif;
          int i;

          /* On comencen les dades dels BMP */
          for(i=0;i<n;i++)  {
                    b[i]=img[i].buf;
                    b[i]+=getValor(BFOFFBITS_S,b[i]+BFOFFBITS);
          }


          /* Primer octet a comparar */
          oct=0;

          /* Inicialitzem el generador de nombres aleatoris */
          srand((unsigned)time(NULL));

          /* Hem comparar un total de BIWIDTH*BIHEIGHT pixels */
          for(pix=0;pix<getValor(BIWIDTH_S,img[0].buf+BIWIDTH)*getValor(BIHEIGHT_S,img[0].b
uf+BIHEIGHT);pix++)  {

                    /* Son diferents els pixels? */
                    pixDif=0;
                    for(i=1;i<n;i++)  {
                              if((b[0][oct]!=b[i][oct]))  {
                                        pixDif=1;
                                        break;
                              }
                              if((b[0][oct+1]!=b[i][oct+1]))  {
                                        pixDif=1;
                                        break;
                              }
```

```
                    if((b[0][oct+2]!=b[i][oct+2]))  {
                              pixDif=1;
                              break;
                    }

          }


          /* Si hi ha hagut diferencies, confabulem */
          if(pixDif!=0)  {

                    /* Prenem un bit aleatori per modificar el bit de
                       menys pes de cada octet del pixel */
                    bit=0+(int)(2.0*rand()/(RAND_MAX+1.0));

                    /* Confabulem el bit de menys pes de cada octet
                       del pixel */

                    /* Octet R */
                    b[0][oct]=b[0][oct]^bit;

                    /* Octet G */
                    b[0][oct+1]=b[0][oct+1]^bit;

                    /* Octet B */
                    b[0][oct+2]=b[0][oct+2]^bit;

          }

          /* Passem al pixel seguent */
          oct+=3;

     }

     /* Fet */
     return 0;
}
```

```
/*****************************************************************************
 bmp.h

 TREBALL FI DE CARRERA - PROTECCIO DEL COPYRIGHT

 Jordi Trubat Dalmau, jtrubat@uoc.edu
*****************************************************************************/

/* Valors estandard BMP */
#define STD_BFTYPE 19778                /* Enter equivalent a 'BM' */
#define STD_BICOMPRESSION 0                     /* Cap compressio */
#define STD_BIBITCOUNT 24               /* 24 bits per pixel */
#define STD_MIDACAP 54                          /* Mida en octets de les capcaleres */

/* Codis d'error i missatges associats del sistema */
#define E_SYS_NPARAMETRESINC 1
#define E_SYS_NPARAMETRESINC_M "Nombre de parametres incorrecte\n"
#define E_SYS_PARAMETREINC 2
#define E_SYS_PARAMETREINC_M "Parametre incorrecte\n"
#define E_SYS_EXTINC 4
#define E_SYS_EXTINC_M "Extensio del fitxer incorrecta\n"
#define E_SYS_NOMEM 5
#define E_SYS_NOMEM_M "Memoria insuficient\n"
#define E_SYS_FTXDES 6
#define E_SYS_FTXDES_M "No es troba el fitxer\n"
#define E_SYS_FTXINOBRIR 7
#define E_SYS_FTXINOBRIR_M "No s'ha pogut obrir el fitxer d'entrada\n"
#define E_SYS_FTXINLLEGIR 8
#define E_SYS_FTXINLLEGIR_M "No s'ha pogut llegir el fitxer d'entrada\n"
#define E_SYS_FTXINTANCAR 9
#define E_SYS_FTXINTANCAR_M "No s'ha pogut tancar el fitxer d'entrada\n"
#define E_SYS_FTXOUTCREAR 10
#define E_SYS_FTXOUTCREAR_M "No s'ha pogut guardar el fitxer de sortida\n"
#define E_SYS_FTXOUTESCRIURE 11
#define E_SYS_FTXOUTESCRIURE_M "No s'ha pogut escriure el fitxer de sortida\n"
#define E_SYS_FTXOUTTANCAR 12
#define E_SYS_FTXOUTTANCAR_M "No s'ha pogut tancar el fitxer de sortida\n"

/* Codis d'error i missatges associats relatius a una imatge BMP */
#define E_BMP_TIPUS 100
#define E_BMP_TIPUS_M "No es una imatge BMP valida\n"
#define E_BMP_CMPR 101
#define E_BMP_CMPR_M "Imatge BMP comprimida\n"
#define E_BMP_BITS 102
#define E_BMP_BITS_M "No es una imatge BMP de 24 bits\n"
#define E_BMP_NOQUAD 103
#define E_BMP_NOQUAD_M "Imatge BMP no quadrada\n"
#define E_BMP_DIFERENTS 104
#define E_BMP_DIFERENTS_M "No es pot comparar les dues imatges\n"

/* Missatge d'error per defecte */
#define E_DESCONEGUT_M "Error desconegut\n"

/* Definicio de tipus octet */
typedef unsigned char octet;
```

```
/* Offsets dels diferents camps (i les seves mides) de les capcaleres d'un
   fitxer BMP. Notacio Windows:-) */
#define BFTYPE 0                    /* Tipus d'imatge */
#define BFTYPE_S 2
#define BFSIZE 2                    /* Mida del fitxer en octets */
#define BFSIZE_S 4
#define BFRESERVED1 6               /* Camps reservats */
#define BFRESERVED1_S 2
#define BFRESERVED2 8
#define BFRESERVED2_S 2
#define BFOFFBITS 10                /* Offset on comencen les dades */
#define BFOFFBITS_S 4
#define BISIZE 14                   /* Mida de la capcalera */
#define BISIZE_S 4
#define BIWIDTH 18                  /* Amplada de la imatge en pixels */
#define BIWIDTH_S 4
#define BIHEIGHT 22                 /* Altura de la imatge en pixels */
#define BIHEIGHT_S 4
#define BIPLANES 26                 /* Plans de colors */
#define BIPLANES_S 2
#define BIBITCOUNT 28               /* Nombre de bits */
#define BIBITCOUNT_S 2
#define BICOMPRESSION 30            /* Compressio */
#define BICOMPRESSION_S 4
#define BISIZEIMAGE 34              /* Mida de la imatge en octets */
#define BISIZEIMAGE_S 4
#define BIXPELSPERMETER 38  /* Pixels horitzontals per metre */
#define BIXPELSPERMETER_S 4
#define BIYPELSPERMETER 42  /* Pixels verticals per metre */
#define BIYPELSPERMETER_S 4
#define BICLRUSED 46                /* Nombre de colors */
#define BICLRUSED_S 4
#define BICLRIMPORTANT 50   /* Nombre de colors importants */
#define BICLRIMPORTANT_S 4

/* Definicio de tipus BMP */
typedef struct BMP
{
        char *nom;                  /* Nom del fixter BMP */
        octet *buf;                 /* Punter al buffer de memoria */
} bmp;

/*** Funcions ***/

/* Llegeix un valor enter a llocs de memoria */
int getValor(int, octet *);

/* Escriu un valor enter a llocs de memoria */
int setValor(int, octet *, unsigned int);

/* Comprova l'estructura d'un fitxer BMP */
int chkBmp(bmp);

/* Imprimeix el valor dels camps i les dades d'un BMP */
int infoBmp(bmp);
```

```
/* Carrega a la memoria un fitxer BMP */
octet *loadBmp(unsigned long *, char *);

/* Guarda al disc una imatge BMP */
int saveBmp(bmp);

/* Compara dues imatges BMP */
int cmpBmp(bmp, bmp, unsigned int *, unsigned int *);

/* Reflecteix hortizontalment una imatge BMP */
int invhBmp(bmp *);

/* Reflecteix verticalment una imatge BMP */
int invvBmp(bmp *);

/* Escala una imatge BMP */
bmp *resBmp(bmp, unsigned int);

/* Imprimeix un missatge d'error */
int errBmp(int);
```

```
/*****************************************************************************
 bmp.c

 Modul de rutines de tractament d'images BMP

 TREBALL FI DE CARRERA - PROTECCIO DEL COPYRIGHT

 Jordi Trubat Dalmau, jtrubat@uoc.edu
*****************************************************************************/

#include<stdio.h>
#include<stdlib.h>
#include<sys/stat.h>
#include<bmp.h>


/*****************************************************************************
 getValor

 Calcula el valor enter positiu representat per un conjunt d'octets
 consecutius de memoria

 descripcio:

        La funcio retorna en un enter el valor representat per un conjunt
        d'octets de memoria consecutius. Els parametres a passar son el
        nombre d'octets consecutius que volem llegir i el punter al primer
        octet. La funcio retorna un enter amb el valor guardat en la cadena
        d'octets seguint l'esquema little-endian. Se sobreenten que el
        nombre d'octets consecutius a llegir no pot excedir el nombre
        d'octets amb el que es guarda un enter (4 octets en linux). A la
        practica l'us d'aquesta funcio es restrigeix a la lectura de 2 o 4
        octets. En qualsevol cas la funcio retorna sempre el valor 0.

 entrada:

        n:      nombre d'octets
        b:    punter al primer octet

 retorna:

        el valor enter
*****************************************************************************/

int getValor(int n, octet *b)
{

        unsigned int t=0; /* Temporal */
        unsigned int valor;         /* Valor resultant */
        int i;                      /* Comptador */

        /* Inicialitzem valor al primer octet */
        valor=(unsigned int) b[0];
```

```
	/* Anem sumant consecutivament la resta d'octets */
	for(i=1;i<n;i++)  {
		b++;		/* Passem a l'octet seguent */
		t=*b;		/* Fem una copia */
		t=t<<8*i;	/* Correm els bits tants octets a l'esquerra
				   com calgui */
		valor+=t;	/* Actualitzem el valor resultant */
	}

	/* Retornem el valor resultant */
	return valor;

}


/*****************************************************************************
 setValor

 Copia un valor enter positiu a una posicio de memoria

 descripcio:

	La funcio escriu el valor d'un enter a un conjunt d'octets de memoria
	consecutius. Els parametres a passar son el nombre d'octets consecutius
	que volem escriure, el punter al primer	octet i l'enter. La funcio
	escriu els octets seguint l'esquema little-endian. Se sobreenten que el
	nombre d'octets consecutius a escriure no excedeix el nombre d'octets
	amb el que es guarda un enter (4 octets en linux). A la practica l'us
	d'aquesta funcio es restrigeix a l'escriptura de 2 o 4 octets. La
	funcio retorna sempre el valor 0.

 entrada:

	n:	nombre d'octets
	b:	punter al primer octet
	a:	enter

 retorna:

	0
*****************************************************************************/

int setValor(int n, octet *b, unsigned int a)
{

	int i;			/* Comptador */

	/* Anem calculant el valor de cada octet */
	for(i=0;i<n;i++)  {
		b[i]=(octet) a;		/* Valor que tindra aquest octet */
		a=a>>8;		/* Correm els bits del valor enter tants
				   octets a la dreta com calgui */
	}

	/* Fet */
	return 0;

}
```

```
/*****************************************************************************
 loadBmp

 Carrega a la memoria el fitxer BMP

 descripcio:

            Carrega  ala memoria un fitxer que (suposadament) conte una imatge
            tipus BMP. Es a dir, fa lloc a la memoria mitjancant els parametres
            mida que especifica la mida del fitxer i el nom del fitxer. Si la
            funcio reexeix, es a dir, si el fitxer realment existeix, si s'ha
            pogut crear el buffer i s'han dut amb exit les operacions i/o
            associades a la lectura d'un fitxer, es retona un punter tipus octet
            que assenyala l'adreca del buffer.

 entrada:

            mida:     mida en octets del fitxer
            fitxer: nom del fitxer

 retorna:

            punter al primer octet de la memoria on s'ha carregat la imatge, o
            NULL en cas d'error
*****************************************************************************/

octet *loadBmp(unsigned long *mida, char *fitxer)
{

            struct stat s;                  /* Estadistiques del fitxer */
            FILE *f;                        /* Descriptor del fitxer */
            octet *buf;                     /* Punter al buffer */

            /* Mida del fitxer a llegir */
            if(stat(fitxer, &s)<0)
            {
                        errBmp(E_SYS_FTXDES);
                        return NULL;
            }
            *mida=s.st_size;

            /* Fem lloc per llegir el fitxer */
            if((buf=(octet*)malloc(*mida))==NULL && *mida!=0)
            {
                        errBmp(E_SYS_NOMEM);
                        return NULL;
            }

            /* Obrim el fitxer */
            if((f=fopen(fitxer, "r"))==NULL)
            {
                        errBmp(E_SYS_FTXINOBRIR);
                        return NULL;
            }
```

```
	/* Llegim el fitxer */
	if(fread(buf, sizeof *buf, *mida, f)<s.st_size)
	{
		errBmp(E_SYS_FTXINLLEGIR);
		return NULL;
	}

	/* Tanquem el fitxer */
	if(fclose(f)!=0)
	{
		errBmp(E_SYS_FTXINTANCAR);
		return NULL;
	}

	/* Retornem l'adreca del buffer */
	return buf;

}


/*****************************************************************************
 chkBmp

 Comprova que l'estructura del fitxer BMP sigui correcta

 descripcio:

	La funcio comprova que el contingut d'un buffer representi una imatge
	BMP valida amb algunes restriccions addicionals. En concret, es
	comprova que sigui una imatge BMP (contingut del camp bfType), que
	sigui quadrada (camps biwidth i biheight iguals), que el nombre de
	bits per pixel sigui 24 (camp biBitCount) i que no estigui comprimida
	(camp biCompression). Si es compleixen aquests requisits es retorna el
	valor zero, en cas contrari -1.

 entrada:

	img:	punter a una instancia de tipus BMP

 retorna:

	0 si es tracta d'una imatge BMP de 24 bits, quadrada i sense comprimir
	-1 en cas contrari
*****************************************************************************/

int chkBmp(bmp img)
{

	/* Tipus d'imatge */
	if(getValor(BFTYPE_S, img.buf+BFTYPE)!=STD_BFTYPE)  {
		errBmp(E_BMP_TIPUS);
		return -1;
	}
```

```
	/* Amplada i altura de la imatge en pixels */
	if(getValor(BIWIDTH_S,img.buf+BIWIDTH)!=getValor(BIHEIGHT_S,img.buf+BIHEIGHT))
{
		errBmp(E_BMP_NOQUAD);
		return -1;
	}

	/* Nombre de bits per pixel */
	if(getValor(BIBITCOUNT_S, img.buf+BIBITCOUNT)!=STD_BIBITCOUNT)  {
		errBmp(E_BMP_BITS);
		return -1;
	}

	/* Tipus de compressio */
	if(getValor(BICOMPRESSION_S, img.buf+BICOMPRESSION)!=STD_BICOMPRESSION)  {
		errBmp(E_BMP_CMPR);
		return -1;
	}

	/* Fet */
	return 0;

}


/*****************************************************************************
 saveBmp

 Guarda al disc una imatge BMP

 descripcio:

	La funcio guarda al disc el contingut d'un buffer que conte una imatge
	tipus BMP. El nom amb el que es guarda el fitxer es el que conte el
	camp nom de l'estructura de dades BMP. Si l'operacio reeixeeix, es a
	dir, si s'ha dut amb exit les operacions i/o associades a la creacio
	del fitxer es retorna el valor 0; -1 en cas contrari.

 entrada:

	img:	punter a una instancia de tipus BMP

 retorna:

	0 si ha reeixit
	-1 en cas d'error
*****************************************************************************/

int saveBmp(bmp img)
{

	FILE *f;		/* Descriptor del fitxer on guardem el BMP */
	unsigned int mida;	/* Mida del fitxer */
```

```
	/* Creem el fitxer */
	if((f=fopen(img.nom, "w"))==NULL)
	{
		errBmp(E_SYS_FTXOUTCREAR);
		return -1;
	}

	/* Escrivim el fitxer */
	mida=getValor(BFSIZE_S, img.buf+BFSIZE);
	if(fwrite(img.buf,sizeof *img.buf,mida,f)<mida)
	{
		errBmp(E_SYS_FTXOUTESCRIURE);
		return -1;
	}

	/* Tanquem el fitxer */
	if(fclose(f)!=0)
	{
		errBmp(E_SYS_FTXOUTTANCAR);
		return -1;
	}

	return 0;

}
```

```
/*****************************************************************************
 invhBmp

 Inverteix horitzontalment els pixels d'una image BMP de 24 bits no comprimida

 descripcio:

        La funcio inverteix horitzontalment els pixels que formen un BMP de
        24 bits sense comprimir. La inversio te lloc en el mateix buffer i,
        per tant, la imatge original es perd. La capcalera no es modifica.

 entrada:

        img:    punter a una instancia de tipus BMP

 sortida:

        img:    punter a la instancia amb els pixels invertits horitzontalment

 retorna:

        0
*****************************************************************************/

int invhBmp(bmp *img)
{

        octet *b;           /* On comencen les dades del BMP */
        int i,j;            /* Comptadors de fila i columna */
        octet t;            /* Temporal per a un octet */
        unsigned int h,w;/* Altura i amplada de la imatge */

        /* Buffer del BMP */
        b=img->buf;

        /* Altura i amplada de la imatge */
        h=getValor(BIHEIGHT_S,b+BIHEIGHT);
        w=getValor(BIWIDTH_S,b+BIWIDTH);

        /* On comencen les dades del BMP */
        b+=getValor(BFOFFBITS_S,b+BFOFFBITS);

        /* Per a cada fila de pixels */
        for(i=0;i<h;i++)  {

                /* Per a la meitat de columnes de pixels */
                for(j=0;j<w/2;j++)  {

                        /* Intercanviem l'octet R */
                        t=b[3*(i*w+j)];
                        b[3*(i*w+j)]=b[3*(i*w+w-j-1)];
                        b[3*(i*w+w-j-1)]=t;

                        /* Intercanviem l'octet G */
                        t=b[3*(i*w+j)+1];
                        b[3*(i*w+j)+1]=b[3*(i*w+w-j-1)+1];
                        b[3*(i*w+w-j-1)+1]=t;

                        /* Intercanviem l'octet B */
```

```
                        t=b[3*(i*w+j)+2];
                        b[3*(i*w+j)+2]=b[3*(i*w+w-j-1)+2];
                        b[3*(i*w+w-j-1)+2]=t;
                }
        }

        /* Fet */
        return 0;
}


/*****************************************************************************
 invvBmp

 Inverteix verticalment els pixels d'una image BMP de 24 bits no comprimida

 descripcio:

        La funcio inverteix verticalment els pixels que formen un BMP de 24
        bits sense comprimir. La inversio te lloc en el mateix buffer i, per
        tant, la imatge original es perd. La capcalera no es modifica.


 entrada:

        img:    punter a una instancia de tipus BMP

 sortida:

        img:    punter a la instancia amb els pixels invertits verticalment

 retorna:

        0
*****************************************************************************/

int invvBmp(bmp *img)
{

        octet *b;           /* On comencen les dades del BMP */
        int i,j;            /* Comptadors de fila i columna */
        octet t;            /* Temporal per a un octet */
        unsigned int h,w;/* Altura i amplada de la imatge */

        /* Buffer del BMP */
        b=img->buf;

        /* Altura i amplada de la imatge */
        h=getValor(BIHEIGHT_S,b+BIHEIGHT);
        w=getValor(BIWIDTH_S,b+BIWIDTH);

        /* On comencen les dades del BMP */
        b+=getValor(BFOFFBITS_S,b+BFOFFBITS);

        /* Per a la meitat de files de pixels */
        for(i=0;i<h/2;i++)  {

                /* Per a cada columna de pixels */
                for(j=0;j<w;j++)  {
```

```
                        /* Intercanviem l'octet R */
                        t=b[3*(i*w+j)];
                        b[3*(i*w+j)]=b[3*((h-i-1)*w+j)];
                        b[3*((h-i-1)*w+j)]=t;

                        /* Intercanviem l'octet G */
                        t=b[3*(i*w+j)+1];
                        b[3*(i*w+j)+1]=b[3*((h-i-1)*w+j)+1];
                        b[3*((h-i-1)*w+j)+1]=t;

                        /* Intercanviem l'octet B */
                        t=b[3*(i*w+j)+2];
                        b[3*(i*w+j)+2]=b[3*((h-i-1)*w+j)+2];
                        b[3*((h-i-1)*w+j)+2]=t;
                }
        }

        /* Fet */
        return 0;
}



/*****************************************************************************
 errBmp

 Imprimeix al canal estandard d'error el missatge d'error corresponent al
 codi passat com a parametre

 descripcio:

        L'objectiu de la funcio es mantenir informat a l'usuari o al
        desenvolupador d'aquest programari de l'origen dels errors o
        excepcions de programari que puguin sorgir durant la seva execucio.
        La funcio imprimeix al canal estandard d'error un missatge
        explicatiu a l'error o excepcio produida en algun punt del programa.
        El missatge va lligat amb el codi d'error passat com a parametre.
        Els codis i els missatges estan definits al fitxer bmp.h.

 entrada:

        codi:   codi d'error

 retorna:

        0
*****************************************************************************/

int errBmp(int codi)
{
        switch(codi)
        {
        case E_SYS_NPARAMETRESINC:
                fprintf(stderr, E_SYS_NPARAMETRESINC_M);
                break;

        case E_SYS_PARAMETREINC:
                fprintf(stderr, E_SYS_PARAMETREINC_M);
                break;
```

```
case E_SYS_EXTINC:
        fprintf(stderr, E_SYS_EXTINC_M);
        break;

case E_SYS_NOMEM:
        fprintf(stderr, E_SYS_NOMEM_M);
        break;

case E_SYS_FTXDES:
        fprintf(stderr, E_SYS_FTXDES_M);
        break;

case E_SYS_FTXINOBRIR:
        fprintf(stderr, E_SYS_FTXINOBRIR_M);
        break;

case E_SYS_FTXINLLEGIR:
        fprintf(stderr, E_SYS_FTXINLLEGIR_M);
        break;




case E_SYS_FTXINTANCAR:
        fprintf(stderr, E_SYS_FTXINTANCAR_M);
        break;

case E_SYS_FTXOUTCREAR:
        fprintf(stderr, E_SYS_FTXOUTCREAR_M);
        break;

case E_SYS_FTXOUTESCRIURE:
        fprintf(stderr, E_SYS_FTXOUTESCRIURE_M);
        break;

case E_SYS_FTXOUTTANCAR:
        fprintf(stderr, E_SYS_FTXOUTTANCAR_M);
        break;

case E_BMP_TIPUS:
        fprintf(stderr, E_BMP_TIPUS_M);
        break;

case E_BMP_CMPR:
        fprintf(stderr, E_BMP_CMPR_M);
        break;

case E_BMP_NOQUAD:
        fprintf(stderr, E_BMP_NOQUAD_M);
        break;

case E_BMP_BITS:
        fprintf(stderr, E_BMP_BITS_M);
        break;

case E_BMP_DIFERENTS:
        fprintf(stderr, E_BMP_DIFERENTS_M);
        break;
```

```
	/* Si no es cap dels anteriors, imprimim un "error desconegut"*/
	default:
		fprintf(stderr, E_DESCONEGUT_M);

	}

	return 0;
}
```

```
/*****************************************************************************
 infobmp.c

 TREBALL FI DE CARRERA - PROTECCIO DEL COPYRIGHT

 Jordi Trubat Dalmau, jtrubat@uoc.edu
*****************************************************************************/

#include<stdio.h>
#include<sys/stat.h>
#include<string.h>
#include<bmp.h>


/*****************************************************************************
 nom:

        infobmp - Imprimeix el contingut d'una imatge tipus BMP

 sinopsi:

        infobmp imatge

 descripcio:

        El programa volca al canal de sortida estandard tota la informacio
        (formatejada) relativa a un fitxer BMP quadrat, de 24 bits no
        comprimit. S'imprimeix (en decimal) els valors de tots els camps de les
        capcaleres i els valors dels tres octets RGB de cada pixel. Els valors
        dels pixels s'imprimeixen tot guardant la logica de files i columnes
        de la imatge. Tenint en compte les mides tipiques d'una imatge, es
        recomana redireccionar la sortida a un fitxer.

*****************************************************************************/

int main(int argc, char **argv)
{

        char *f;                /* Descriptor dels fitxer BMP */
        unsigned long mida;     /* Mida del fitxer BMP */
        octet *buf;             /* Punters al buffer */
        bmp img;                 /* Instancia BMP */


        /* Nombre de parametres correcte? */
        if(argc!=2)  {
                errBmp(E_SYS_NPARAMETRESINC);
                return -1;
        }

        /* Nom de la imatge BMP */
        f=argv[1];
        if((strstr(f,"bmp")==NULL)&&(strstr(f,"BMP")==NULL)) {
                errBmp(E_SYS_EXTINC);
                return -1;
        }
```

```
	/* Carreguem la imatge a la memoria */
	if((buf=loadBmp(&mida, f))==NULL)  {
		return -1;
	}

	/* Omplim els camps de l'estructura BMP de la imatge */
	img.nom=f;
	img.buf=buf;

	/* Comprovem l'estructura de la imatge */
	if(chkBmp(img)!=0)  {
		return -1;
	}

	/* Imprimim els valors dels camps i les dades de la imatge */
	infoBmp(img);

	/* Fet */
	return 0;

}


/*****************************************************************************
 infoBmp

 Imprimeix els valors dels camps de l'estructura BMP

 descripcio:

	La funcio volca al canal de sortida estandard tota la informacio
	(formatejada) relativa a un fitxer BMP de 24 bits no comprimit. Es a
	dir, s'imprimeix (en decimal) els valors de tots els camps de les
	capcaleres i els valors dels tres octets RGB de cada pixel. Els valors
	dels pixels s'imprimeixen tot guardant la logica de files i columnes
	de la imatge. Tenint en compte les mides tipiques d'una imatge, es
	recomana redireccionar la sortida a un fitxer.

 entrada:

	img:	punter a una instancia de tipus BMP

 retorna:

	0
*****************************************************************************/

int infoBmp(bmp img)
{

	octet *b;		/* Punter a l'octet on comencen les dades del BMP */
	int i,j;		/* Comptador */
	unsigned int oct; /* Index per a les dades del BMP */
```

```
	/* Nom del BMP */
	printf("NOM:\n");
	printf("%s\n", img.nom);
	printf("\n");

	/* Guardem l'adreca del buffer */
	b=img.buf;

	/* Capcalera del BMP */
	printf("CAP:\n");
	printf("bfType:	%d\n", getValor(BFTYPE_S,b+BFTYPE));
	printf("bfSize:	%d\n", getValor(BFSIZE_S,b+BFSIZE));
	printf("bfReserved1:	%d\n", getValor(BFRESERVED1_S,b+BFRESERVED1));
	printf("bfReserved2:	%d\n", getValor(BFRESERVED2_S,b+BFRESERVED2));
	printf("bfOffBits:	%d\n", getValor(BFOFFBITS_S,b+BFOFFBITS));
	printf("\n");

	/* Capcalera d'informacio del BMP */
	printf("CAP INFORMACIO:\n");
	printf("biSize:	%d\n", getValor(BISIZE_S,b+BISIZE));
	printf("biWidth:	%d\n", getValor(BIWIDTH_S,b+BIWIDTH));
	printf("biHeight:	%d\n", getValor(BIHEIGHT_S,b+BIHEIGHT));
	printf("biPlanes:	%d\n", getValor(BIPLANES_S,b+BIPLANES));
	printf("biBitCount:	%d\n", getValor(BIBITCOUNT_S,b+BIBITCOUNT));
	printf("biCompression:	%d\n", getValor(BICOMPRESSION_S,b+BICOMPRESSION));
	printf("biSizeImage:	%d\n", getValor(BISIZEIMAGE_S,b+BISIZEIMAGE));
	printf("biXPelsPerMeter: %d\n",
getValor(BIXPELSPERMETER_S,b+BIXPELSPERMETER));
	printf("biYPelsPerMeter: %d\n",
getValor(BIYPELSPERMETER_S,b+BIYPELSPERMETER));
	printf("biClrUsed:	%d\n", getValor(BICLRUSED_S,b+BICLRUSED));
	printf("biClrImportant:	%d\n", getValor(BICLRIMPORTANT_S,b+BICLRIMPORTANT));
	printf("\n");

	/* Dades del BMP */
	printf("DADES:\n");

	/* On comencen les dades */
	b+=getValor(BFOFFBITS_S,b+BFOFFBITS);

	/* Primer octet */
	oct=0;

	/* Per a cada fila de pixels */
	for(i=0;i<getValor(BIHEIGHT_S,img.buf+BIHEIGHT);i++) {

		/* Per a cada columna de pixels */
		for(j=0;j<getValor(BIWIDTH_S,img.buf+BIWIDTH);j++) {

			/* Octet R */
			printf("%d ", b[oct++]);

			/* Octet G */
			printf("%d ", b[oct++]);
```

```
                        /* Octet B */
                        printf("%d-", b[oct++]);
                }

                /* Pasem a la fila seguent de pixels */
                printf("\n");

        }
        /* Acabem amb elegancia */
        printf("\n");

        /* Fet */
        return 0;


}



/*****************************************************************************
 cmpbmp.c

 TREBALL FI DE CARRERA - PROTECCIO DEL COPYRIGHT

 Jordi Trubat Dalmau, jtrubat@uoc.edu
*****************************************************************************/

#include<stdio.h>
#include<sys/stat.h>
#include<string.h>
#include<bmp.h>


/*****************************************************************************
 nom:

        cmpbmp - Compara dues imatges tipus BMP

 sinopsi:

        cmpbmp imatge1 imatge2

 descripcio:

        Es compara octet a octet i pixel a pixel dues imatges BMP de 24 bits
        sense comprimir passades com a parametres a la linia de comandes.
        En primer lloc el programa comprova que les dues imatges siguin dues
        images BMP valides amb les restriccions addicionals que siguin
        quadrades, sense comprimir i de 24 bits. A continuacio es comprova
        si les capcaleres son iguals. Si algun dels camps no coincideix,
        el programa retorna un missatge     d'error tot indicant que no s'ha pogut
        fer la comparacio. Si les capcaleres son iguals, es compara un a un
        cadascun dels octets del camps de dades. El programa imprimeix pel
        canal de sortida estandard el nombre d'octets i de pixels en que
        diferexein les imatges i els percentatges relatius als totals.

*****************************************************************************/
```

```
int main(int argc, char **argv)
{

        char *f1, *f2;                  /* Descriptors dels fitxers BMP */
        unsigned long mida1, mida2;     /* Mides dels fitxers BMP */
        octet *buf1, *buf2;             /* Punters als buffers */
        bmp img1, img2;                 /* Instancies  BMP */
        unsigned int noct;              /* Nombre d'octets diferents */
        unsigned int npix;              /* Nombre de pixels diferents */


        /* Nombre de parametres correcte? */
        if(argc!=3)  {
                errBmp(E_SYS_NPARAMETRESINC);
                return -1;
        }

        /* Nom de la primera imatge BMP */
        f1=argv[1];
        if((strstr(f1,"bmp")==NULL)&&(strstr(f1,"BMP")==NULL)) {
                errBmp(E_SYS_EXTINC);
                return -1;
        }

        /* Nom de la segona imatge BMP */
        f2=argv[2];
        if((strstr(f2,"bmp")==NULL)&&(strstr(f2,"BMP")==NULL)) {
                errBmp(E_SYS_EXTINC);
                return -1;
        }

        /* Carreguem la primera imatge a la memoria */
        if((buf1=loadBmp(&mida1, f1))==NULL)  {
                return -1;
        }

        /* Carreguem la segona imatge a la memoria */
        if((buf2=loadBmp(&mida2, f2))==NULL)  {
                return -1;
        }

        /* Omplim els camps de l'estructura BMP de la primera imatge */
        img1.nom=f1;
        img1.buf=buf1;

        /* Omplim els camps de l'estructura BMP de la segona imatge */
        img2.nom=f2;
        img2.buf=buf2;

        /* Comprovem l'estructura de la primera imatge */
        if(chkBmp(img1)!=0)  {
                return -1;
        }

        /* Comprovem l'estructura de la segona imatge */
        if(chkBmp(img2)!=0)  {
                return -1;
        }
```

```
	/* Comparem les imatges */
	if(cmpBmp(img1,img2,&noct,&npix)!=0)  {
		return -1;
	}
	printf("Les imatges difereixen en: ");
	printf("%d octets (%.2f%%), ",noct,(float) 100*noct/getValor(BISIZEIMAGE_S,
img1.buf+BISIZEIMAGE));
	printf("%d pixels (%.2f%%)\n",npix,(float) 300*npix/getValor(BISIZEIMAGE_S,
img1.buf+BISIZEIMAGE));

	/* Fet */
	return 0;

}


/*****************************************************************************
 cmpBmp

 Compara dues imatges tipus BMP

 descripcio:

	Es compara octet a octet i pixel a pixel dues imatges BMP de 24 bits.
	En primer lloc es compara tots els camps de les dues capcaleres. Si
	algun d'aquests camps no coincideix, la funcio retorna el valor -1
	indicant que no s'ha pogut fer la comparacio. Si les capcaleres son
	iguals, es compara dos a dos cadascun dels octets del camp de dades.
	Es mantenen dos comptadors, un pel nombre d'octets en que difereixen
	les imatges, i un altre pel nombre de pixels diferents. Aquests
	comptadors son els valors retornats per la funcio en forma de
	parametres.

 entrada:

	img1:	primera instancia tipus BMP
	img2:	segona instancia tipus BMP

 sortida:

	noct:	nombre d'octets en que difereixen les imatges
	npix:	nombre de pixels en que difereixen les imatges

 retorna:
	0	si s'ha pogut fer la comparacio
	-1	si no s'ha pogut fer la comparacio, p. ex. si les imatges
		no son de la mateixa mida
*****************************************************************************/

int cmpBmp(bmp img1, bmp img2, unsigned int *noct, unsigned int *npix)
{

	octet *b1,*b2;		/* Punters als buffers de les dues imatges */
	unsigned int i;		/* Comptador de pixels */
	unsigned int t;		/* Temporal */
	unsigned int oct;	/* Index per als octets de la imatge */
```

```
	/* Comparem les capcaleres. Si no coincideixen, no es pot
	   comparar les imatges */
	if(memcmp(img1.buf,img2.buf,STD_MIDACAP)!=0)  {
		errBmp(E_BMP_DIFERENTS);
		return -1;
	}

	/* On comencen les dades del primer BMP */
	b1=img1.buf;
	b1+=getValor(BFOFFBITS_S,b1+BFOFFBITS);

	/* On comencen les dades del segon BMP */
	b2=img2.buf;
	b2+=getValor(BFOFFBITS_S,b2+BFOFFBITS);

	/* Inicialitzem el nombre d'octets diferents a zero */
	*noct=0;

	/* Inicialitzem el nombre de pixels diferents a zero */
	*npix=0;

	/* Primer octet a comparar */
	oct=0;

	/* Hem comparar un total de BIWIDTH*BIHEIGHT pixels */
	for(i=0;i<getValor(BIWIDTH_S,img1.buf+BIWIDTH)*getValor(BIHEIGHT_S,img1.buf+BIH
EIGHT);i++)  {

		/* Fem una copia de noct */
		t=*noct;

		/* Octets R */
		if(b1[oct]!=b2[oct])  (*noct)++;
		oct++;

		/* Octets G */
		if(b1[oct]!=b2[oct])  (*noct)++;
		oct++;

		/* Octets B */
		if(b1[oct]!=b2[oct])  (*noct)++;
		oct++;

		/* Si noct ha canviat, el pixel era diferent */
		if(t!=*noct) (*npix)++;

	}

	/* Fet */
	return 0;
}
```

```
################################################################################
# Makefile
#
# TREBALL FI DE CARRERA - PROTECCIO DEL COPYRIGHT
#
# Jordi Trubat Dalmau, jtrubat@uoc.edu
################################################################################

# Makefile per compilar marca, confabula i algunes rutines auxiliars

# Compilador
CC = gcc

# Flags
CFLAGS = -Wall -O3 -m486

# A compilar
all:	marca confabula cmpbmp infobmp

marca: marca.o pMarca.o eMarca.o hamMarca.o bdMarca.o errMarca.o bmp.o
	$(CC) $(CFLAGS) -o marca marca.c pMarca.c eMarca.c hamMarca.c bdMarca.c errMarca.c
bmp.c

confabula: confabula.o errMarca.o bmp.o
	$(CC) $(CFLAGS) -o confabula confabula.c errMarca.c bmp.c

cmpbmp: cmpbmp.o
	$(CC) $(CFLAGS) -o cmpbmp cmpbmp.c bmp.c

infobmp: infobmp.o
	$(CC) $(CFLAGS) -o infobmp infobmp.c bmp.c

marca.o: marca.c marca.h
	$(CC) $(CFLAGS) -c marca.c

pMarca.o: pMarca.c marca.h
	$(CC) $(CFLAGS) -c pMarca.c

eMarca.o: eMarca.c marca.h
	$(CC) $(CFLAGS) -c eMarca.c

hamMarca.o: hamMarca.c
	$(CC) $(CFLAGS) -c hamMarca.c

bdMarca.o: bdMarca.c marca.h
	$(CC) $(CFLAGS) -c bdMarca.c

errMarca.o: errMarca.c marca.h
	$(CC) $(CFLAGS) -c errMarca.c

bmp.o: bmp.c bmp.h
	$(CC) $(CFLAGS) -c bmp.c
```

```
#!/bin/bash
#confabula.sh

echo "Script Joc de Proves Confabulacions"
echo "TFC (Proteccio del copyright)"
echo "Jordi Trubat Dalmau, jtrubat@uoc.edu"
echo

# Nombre d'iteracions
ITERS=100

# Nom de les imatges
IMATGE1=lenna
IMATGE2=logo

# Etiqueta de la imatge confabulada
CONF=conf

# Extensio dels fitxers
EXT=bmp

# Funcio per confabular amb 2 compradors i extreure la marca
conf2()
{
	echo "> Creant $ITERS confabulacions de $CLIENT1 i $CLIENT2 i llegint la marca..."
	echo
	local i=1
	while [ "$i" -le $ITERS ]
	do
		confabula $IMATGE.$CLIENT1.$EXT $IMATGE.$CLIENT2.$EXT
		marca $IMATGE.$CONF.$CLIENT1.$CLIENT2.$EXT e $IMATGE.$EXT
		let "i+=1"
	done
	echo
}

# Funcio per confabular amb 3 compradors i extreure la marca
conf3()
{
	echo "> Creant $ITERS confabulacions de $CLIENT1, $CLIENT2 i $CLIENT3 i llegint la
marca..."
	echo
	local i=1
	while [ "$i" -le $ITERS ]
	do
		confabula $IMATGE.$CLIENT1.$EXT $IMATGE.$CLIENT2.$EXT
$IMATGE.$CLIENT3.$EXT
		marca $IMATGE.$CONF.$CLIENT1.$CLIENT2.$CLIENT3.$EXT e
$IMATGE.$EXT
		let "i+=1"
	done
	echo
}
```

```
for IMATGE in $IMATGE1 $IMATGE2
do

	# Marquem la imatge per a cada comprador
	echo "ENCAST I EXTRACCIO DE LA MARCA PER A CLIENTS (IMATGE: $IMATGE)"
	echo
	for CLIENT in "jordi" "anna" "claudia" "antoni" "petra" "joan" "pere"
	do
		echo "> Marcant $IMATGE.$EXT per al client $CLIENT..."
		marca $IMATGE.$EXT m $CLIENT
		echo "Fitxer creat: $(ls $IMATGE.$CLIENT.$EXT)"
		echo "> Extraient la marca de $IMATGE.$CLIENT.$EXT..."
		marca $IMATGE.$CLIENT.$EXT e $IMATGE.$EXT
		echo
	done


	# Fem les confabulacions i extreiem la marca
	echo "CONFABULACIONS DE 2 COMPRADORS I EXTRACCIO DE LA MARCA
(IMATGE: $IMATGE)"
	echo

	CLIENT1=jordi
	CLIENT2=anna
	conf2

	CLIENT1=jordi
	CLIENT2=pere
	conf2

	CLIENT1=claudia
	CLIENT2=petra
	conf2

	CLIENT1=petra
	CLIENT2=antoni
	conf2

	CLIENT1=joan
	CLIENT2=jordi
	conf2

	echo "CONFABULACIONS DE 3 COMPRADORS I EXTRACCIO DE LA MARCA
(IMATGE: $IMATGE)"
	echo

	CLIENT1=jordi
	CLIENT2=anna
	CLIENT3=petra
	conf3

	CLIENT1=pere
	CLIENT2=claudia
	CLIENT3=antoni
	conf3
```

```
        CLIENT1=anna
        CLIENT2=pere


        CLIENT3=joan
        conf3

        CLIENT1=jordi
        CLIENT2=joan
        CLIENT3=anna
        conf3


done


exit
```