DEPARTAMENTO DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico Número 2

Algoritmos y Estructuras de Datos I

Grupo: 07

Integrante	LU	Correo electrónico
Demartino, Francisco	348/14	demartino.francisco@gmail.com
Frachtenberg Goldsmit, Kevin	247/14	kevinfra94@gmail.com
Gomez, Horacio	756/13	horaciogomez.1993@gmail.com
Pondal, Iván	078/14	ivan.pondal@gmail.com

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria - (Pabellón I/Planta Baja)
Intendente Güiraldes 2160 - C1428EGA
Ciudad Autónoma de Buenos Aires - Rep. Argentina
Tel/Fax: (54 11) 4576-3359
http://www.fcen.uba.ar

1. Especificación

Ejercicio 1. problema posicionesMasOscuras($i : Imagen$) = $res : [(\mathbb{Z}, \mathbb{Z})]$\{
asegura : $mismos(res, [(x, y) \mid y \leftarrow [0..alto(i)), x \leftarrow [0..ancho(i)),$
$sumaCanalesPixel(color(i, x, y)) == menorSumaCanales(i)]$);
}

Ejercicio 2. problema top10($g : Galeria$) = $result : [Imagen]$\{
asegura topDiezOMenos: $|result| == min(|imagenes(g)|, 10)$;
asegura estanEnGaleria: $(\forall im \leftarrow result)im \in imagenes(g)$;
asegura sinRepetidos: $(\forall i, j \leftarrow [0..|result|), i \neq j)result[i] \neq result[j]$;
asegura ordenPorVotos: $(\forall i \leftarrow [0..|result|))$
$votos(g, result[i]) == listaVotosOrdenados(g, imagenes(g))[i]$;
}

Ejercicio 3. problema laMasChiquitaConPuntoBlanco($g : Galeria$) = $result : Imagen$\{
requiere : $|listaImagenesConPixelBlanco(imagenes(g))| > 0$;
asegura tienePuntoBlanco: $result \in listaImagenesConPixelBlanco(imagenes(g))$;
asegura esChiquita: $area(result) == minimo($
$[area(i)|i \leftarrow listaImagenesConPixelBlanco(imagenes(g))]$);
}

Ejercicio 4. problema agregarImagen($g : Galeria, i : Imagen$)\{
modifica g;
asegura lasDeAntesEstanConSusVotos: $(\forall j \leftarrow imagenes(pre(g)))$
$j \in imagenes(g) \wedge votos(g, j) == votos(pre(g), j)$;
asegura lasQueEstanSalvoLaQueAgregoVienenDeAntes: $(\forall j \leftarrow imagenes(g), j \neq i)$
$j \in imagenes(pre(g))$;
asegura siAgregoNuevaEntraConCeroVotos: $i \notin imagenes(pre(g)) \rightarrow$
$(i \in imagenes(g) \wedge votos(g, i) == 0)$;
}

Ejercicio 5. problema votar($g : Galeria, i : Imagen$)\{
requiere noSeVotaCualquierCosa: $i \in imagenes(pre(g))$;
modifica g;
asegura noCambianLasImagenes: $mismos(imagenes(g), imagenes(pre(g)))$;
asegura noSeTocanLosVotosDeLosOtros: $(\forall m \leftarrow imagenes(g), m \neq i)\ votos(g, m) ==$
$votos(pre(g), m)$;
asegura elQueSeVotaSumaUno: $votos(g, i) == votos(pre(g), i) + 1$;
}

Ejercicio 6. problema eliminarMasVotada($g : Galeria$)\{
requiere noVacia: $|imagenes(pre(g))| > 0$;
modifica g;
asegura seVaUnaSola: $|imagenes(g)| == |imagenes(pre(g))| - 1$;
asegura laQueSeFueEraGrosa: $(\forall i \leftarrow imagenes(pre(g)),\ i \notin imagenes(g))$
$votos(pre(g), i) == maximo(todosLosVotos(pre(g)))$;
asegura lasQueEstanVienenDeAntesConSusVotos: $(\forall i \leftarrow imagenes(g))$
$i \in imagenes(pre(g)) \wedge votos(g, i) == votos(pre(g), i)$;
}

1.1. Auxiliares

- aux $cuenta(x : T, a : [T]) : \mathbb{Z} = |[1 \mid y \leftarrow a, y == x]|$;
- aux $mismos(a, b : [T]) : Bool = |a| == |b| \wedge (\forall c \leftarrow a) cuenta(c, a) == cuenta(c, b)$;
- aux $minimo(l : [\mathbb{Z}]) : \mathbb{Z} = [x \mid x \leftarrow l, (\forall y \leftarrow l) x \leq y)][0]$;
- aux $maximo(l : [\mathbb{Z}]) : \mathbb{Z} = [x \mid x \leftarrow l, (\forall y \leftarrow l) x \geq y)][0]$;
- aux $sumaCanalesPixel(p : Pixel) : \mathbb{Z} = red(p) + green(p) + blue(p)$;
- aux $listaSumaCanalesPixeles(i : Imagen) : [\mathbb{Z}] =$ $[sumaCanalesPixel(color(i, x, y)) \mid y \leftarrow [0..alto(i)), x \leftarrow [0..ancho(i))]$;
- aux $area(i : Imagen) : \mathbb{Z} = ancho(i) * alto(i)$;
- aux $menorSumaCanales(i : Imagen) : \mathbb{Z} = minimo(listaSumaCanalesPixeles(i))$;
- aux $todosLosVotos(g : Galeria) : [\mathbb{Z}] = [votos(g, i) | i \leftarrow imagenes(g)]$;
- aux $esPixelBlanco(px : Pixel) : Bool = red(px) == green(px) == blue(px) == 255$;
- aux $tienePixelBlanco(i : Imagen) : Bool =$ $alguno([esPixelBlanco(color(i, x, y)) | y \leftarrow [0..alto(i)), x \leftarrow [0..ancho(i))])$;
- aux $listaImagenesConPixelBlanco(imgs : [Imagen]) : [Imagen] = [im \mid im \leftarrow imgs, tienePixelBlanco(im)]$;
- aux $cuentaMasVotos(g : Galeria, imgs : [Imagen], img : Imagen) : \mathbb{Z} =$ $|[1 \mid im \leftarrow imgs, votos(g, im) > votos(g, img)]|$;
- aux $listaVotosOrdenados(g : Galeria, imgs : [Imagen]) : [\mathbb{Z}] =$ $[votos(g, im) \mid i \leftarrow [0..|imgs|), im \leftarrow imgs, cuentaMasVotos(g, imgs, im) == i]$;

2. Demostraciones

2.1. Predicado de abstracción e invariante de representación

```
class GaleriaImagenes {
        public:
                void dividirYAgregar(const Imagen &imagen, int n, int m);
                Imagen laMasChiquitaConPuntoBlanco() const;
                void agregarImagen(const Imagen &imagen);
                void votar(const Imagen &imagen);
                void eliminarMasVotada();
                vector <Imagen> top10() const;
                void guardar(std::ostream& os) const;
                void cargar (std::istream& is);
        private:
                void acomodar();
                bool existeImagen(const Imagen &imagen);
                std::vector<Imagen> imagenes;
                std::vector<int> votos;

        // InvRep(imp : GaleriaImagenes):
        // |imp.imagenes| == |imp.votos| ∧
        // (∀v ← imp.votos) v ≥ 0 ∧
        // (∀i, j ← [0..|imp.imagenes|), i < j)
        // imp.imagenes[i] ≠ imp.imagenes[j] ∧ imp.votos[i] ≤ imp.votos[j]

        // abs(imp : GaleriaImagenes, esp : Galeria):
        // mismos(imp.imagenes, imagenes(esp)) ∧
        // (∀i ← [0..|imp.votos|)) imp.votos[i] == votos(esp, imp.imagenes[i])
};
```

2.2. Preservación del invariante de representación

```
void GaleriaImagenes::eliminarMasVotada() {
        // E0:
        // implica InvRep(this)
        //      por ser un metodo publico, InvRep se asume verdadero
        // implica abs(this, g)
        //      porque InvRep(this) es verdadero
        // vale |imagenes(g)| > 0
        //      por requiere
        // implica |this.imagenes| > 0
        //      por abs(this, g)
        //      ya que vale mismos(this.imagenes, imagenes(g))
        // implica |this.votos| > 0
        //      porque InvRep(this) es verdadero con lo cual
        //      |this.imagenes| == |this.votos|
        // implica P_Aux1: |imagenes(g)| > 0
        this->imagenes.pop_back();
        // E1:
        // vale Q_Aux1:
        // |this.imagenes| == |this.imagenes@E0| - 1 ∧
        // (∀i ← [0..|this.imagenes@E0| - 1)) this.imagenes[i] == this.imagenes@E0[i]
        // vale this.votos == this.votos@E0
        // implica P_Aux2: |this.votos| > 0
        this->votos.pop_back();
        // E2:
        // vale Q_Aux2:
        // |this.votos| == |this.votos@E1| - 1 ∧
        // (∀i ← [0..|this.votos@E1| - 1)) this.votos[i] == this.votos@E1[i]
        // vale this.imagenes == this.imagenes@E1

        // implica |this.imagenes| == |this.imagenes@E0| - 1 ∧ |this.votos| == |this.votos@E0| - 1
        //      por transformacion de estados
        // implica |this.imagenes| == |this.votos|
        //      por implica anterior e InvRep@E0

        // implica (∀i ← [0..|this.votos@E0| - 1)) this.votos[i] == this.votos@E0[i]
        //      por transformacion de estados
        // implica (∀v ← this.votos) v == v@E0
        //      por implica anterior, donde se estaba recorriendo todo el listado
        // implica (∀v ← this.votos) v ≥ 0
        //      por InvRep@E0

        // implica (∀i ← [0..|this.votos@E0| - 1)) this.votos[i] == this.votos@E0[i] ∧
        // (∀j ← [0..|this.imagenes@E0| - 1)) this.imagenes[j] == this.imagenes@E0[j]
        //      por transformacion de estados
        // implica (∀i, j ← [0..|this.imagenes@E0| - 1), i < j)
        // this.imagenes[i] ≠ this.imagenes[j] ∧ this.votos[i] ≤ this.votos[j]
        //      por InvRep@E0
        // implica (∀i, j ← [0..|this.imagenes|), i < j)
        // this.imagenes[i] ≠ this.imagenes[j] ∧ this.votos[i] ≤ this.votos[j]
        //      por transformacion de estados

        // implica InvRep(this)
}
```

2.3. Demostración de correctitud

Ahora que demostramos que se restaura el invariante de representación a la salida del método podemos usar el predicado de abstracción para demostrar que también se cumple la postcondición.

```
// implica |this.imagenes| == |this.imagenes@E0| - 1
//       por transformacion de estados
// implica |imagenes(g)| == |imagenes(pre(g))| - 1
//       por abs(this, g)
//       mismos(this.imagenes, imagenes(g))
//       por abs(pre(this), pre(g))
//       mismos(this.imagenes@E0, imagenes(pre(g))

// implica (∀i ← imagenes(pre(g)), i ∉ imagenes(g))
// votos(pre(g), i) == maximo(todosLosVotos(pre(g)))
//       por InvRep(this), ya que nos aseguraba que las imagenes
//       se encontrasen ordenadas por votos en orden creciente
//       removiendo la ultima en el listado, se removio la mas votada

// implica (∀i ← [0..|this.imagenes@E0| - 1))
// this.imagenes[i] == this.imagenes@E0[i] ∧ this.votos[i] == this.votos@E0[i]
//       por transformacion de estados e InvRep@E0
//       (la longitud de imagenes que es la misma que la de votos)
// implica (∀i ← [0..|imagenes(pre(g))| - 1))
// imagenes(g)[i] == imagenes(pre(g))[i] ∧
// votos(g, imagenes(g)[i]) == votos(pre(g), imagenes(pre(g))[i])
//       por abs(this, g)
//       mismos(this.imagenes, imagenes(g))
//       (∀i ← [0..|this.votos|)) this.votos[i] == votos(g, this.imagenes[i])
//       por abs(pre(this), pre(g))
//       mismos(this.imagenes@E0, imagenes(pre(g))
//       (∀i ← [0..|this.votos@E0|)) this.votos@E0[i] == votos(pre(g), this.imagenes@E0[i])
// implica (∀i ← [0..|imagenes(g)|))
// imagenes(g)[i] == imagenes(pre(g))[i] ∧
// votos(g, imagenes(g)[i]) == votos(pre(g), imagenes(pre(g))[i])
//       por primer asegura |imagenes(g)| == |imagenes(pre(g))| - 1
// implica (∀i ← imagenes(g)) i ∈ imagenes(pre(g)) ∧ votos(g, i) == votos(pre(g), i)
//       por implica anterior, se recorrian todos los valores de imagenes
//       con lo cual se puede escribir con un paratodo
```

3. Código fuente

3.1. pixel.h

```
#ifndef PIXEL_H
#define PIXEL_H

#include <string>
#include <iostream>

using namespace std;

class Pixel {
 public:
  Pixel(int red, int green, int blue);
  Pixel();
  void cambiarPixel(int red, int green, int blue);
  int red() const;
  int green() const;
  int blue() const;
  bool operator==(const Pixel &otro) const;
  bool operator!=(const Pixel &otro) const;

  void guardar(std::ostream& os) const;
  void cargar (std::istream& is);
 private:
  int intensidades[3];
};

#endif  // PIXEL_H
```

3.2. pixel.cpp

```cpp
#include "pixel.h"

Pixel::Pixel(int red, int green, int blue) {
  this->cambiarPixel(red, green, blue);
}

Pixel::Pixel () {
  this->cambiarPixel(0, 0, 0);
}

void Pixel::cambiarPixel(int red, int green, int blue) {
    this->intensidades[0] = red;
    this->intensidades[1] = green;
    this->intensidades[2] = blue;
}

int Pixel::red() const {
  return this->intensidades[0];
}

int Pixel::green() const {
  return this->intensidades[1];
}

int Pixel::blue() const {
  return this->intensidades[2];
}

bool Pixel::operator==(const Pixel &p) const {
   return (
             (this->intensidades[0] == p.red()) &&
             (this->intensidades[1] == p.green()) &&
             (this->intensidades[2] == p.blue())
           );
}

bool Pixel::operator!=(const Pixel &p) const {
   return (
             (this->intensidades[0] != p.red()) ||
             (this->intensidades[1] != p.green()) ||
             (this->intensidades[2] != p.blue())
           );
}

void Pixel::guardar(std::ostream& os) const {
  os  << '('
      << this->intensidades[0]
      << ';'
      << this->intensidades[1]
      << ';'
      << this->intensidades[2]
      << ')';
}


void Pixel::cargar(std::istream& is) {
```

```
57    char c;
58    int r, g, b;
59
60    is >> c; // (
61
62    is >> r;
63
64    is >> c; // ;
65
66    is >> g;
67
68    is >> c; // ;
69
70    is >> b;
71
72    is >> c; // )
73
74    this->cambiarPixel(r,g,b);
75  }
```

3.3. imagen.h

```
#ifndef IMAGEN_H
#define IMAGEN_H

#define BLUR 0
#define ACUARELA 1

#include "pixel.h"

#include <vector>

typedef std::vector<Pixel> Pixel1DContainer;
typedef std::vector<Pixel1DContainer> Pixel2DContainer;


class Imagen {
 public:
  Imagen(int alto_param, int ancho_param);
  Pixel obtenerPixel(int fila, int columna) const;
  void modificarPixel(int fila, int columna, const Pixel &pixel);
  int alto() const;
  int ancho() const;
  vector<pair<int, int> > posicionesMasOscuras() const;
  void blur(int k);
  void acuarela(int k);

  bool operator==(const Imagen &otra) const;

  void guardar(std::ostream& os) const;
  void cargar (std::istream& is);

 private:
  Pixel2DContainer pixels;
  int sumaCanales(int x, int y) const;
  int colorMasOscuro() const;
  int maxPos(const int a[], int desde, int hasta) const;
  void upSort(int a[], int n) const;
  bool kVecinosCompletos(int k, int x, int y) const;
  void aplicarFiltro(int filtro, int k);
  Pixel pixelPromedioKVecinos(int k, int x, int y) const;
  Pixel pixelMedianaKVecinos(int k, int x, int y) const;
};

#endif  // IMAGEN_H
```

3.4. imagen.cpp

```cpp
#include "imagen.h"
#include "pixel.h"

#include <algorithm>
#include <vector>

Imagen::Imagen(int alto_param, int ancho_param) {

    for (int y=0; y<alto_param; y++) {
      Pixel1DContainer fila;
      for (int x=0; x<ancho_param; x++) {
        Pixel p;
        fila.push_back(p);
      }
      this->pixels.push_back(fila);
    }

}

Pixel Imagen::obtenerPixel(int fila, int columna) const {
    return this->pixels.at(fila).at(columna);
}

void Imagen::modificarPixel(int fila, int columna, const Pixel &pixel) {
    this->pixels.at(fila).at(columna).cambiarPixel(
      pixel.red(), pixel.green(), pixel.blue()
    );
}

int Imagen::alto() const {
  return this->pixels.size();
}

int Imagen::ancho() const {
  return this->pixels.at(0).size();
}

vector<pair<int, int> > Imagen::posicionesMasOscuras() const {
  vector<pair<int, int> > pixelesOscuros;
  int x=0; int y=0;
  int colorMasOscuro = this->colorMasOscuro();
  while(x < this->ancho()){
      y=0;
            while(y < this->alto()){
                    if(this->sumaCanales(x,y) == colorMasOscuro) {
                            pixelesOscuros.push_back(pair<int, int>(x,y));
                    }
                    y++;
            }
            x++;
  }
  return pixelesOscuros;
}

void Imagen::blur(int k){
        this->aplicarFiltro(BLUR, k);
```

```
57  }
58
59  void Imagen::acuarela(int k){
60          this->aplicarFiltro(ACUARELA, k);
61  }
62
63  bool Imagen::operator==(const Imagen &otra) const{
64          bool equals=true;
65          if(this->ancho() == otra.ancho() && this->alto()== otra.alto()){
66                  int x=0; int y=0;
67                  while (y < this->alto() && equals){
68                          while(x < this->ancho() && equals){
69                                  if(this->pixels[y][x] != otra.obtenerPixel(y,x)){
70                                          equals=false;
71                                  }
72                                  x++;
73                          }
74                          x=0;
75                          y++;
76                  }
77          }
78          else{
79                  equals=false;
80          }
81          return equals;
82  }
83
84  void Imagen::guardar(std::ostream& os) const {
85
86    os << this->alto() << ' ' << this->ancho() << ' ';
87
88    os << '[';
89
90    for (int y=0; y<this->alto(); y++) {
91        for (int x=0; x<this->ancho(); x++) {
92
93            if ( !(y == 0 && x == 0) )  {
94                os << ',';
95            }
96
97            this->pixels[y][x].guardar(os);
98
99        }
100   }
101
102   os << ']';
103 }
104
105 void Imagen::cargar (std::istream& is) {
106     int alto, ancho;
107     char c;
108
109     this->pixels.clear();
110
111     is >> alto >> ancho;
112
113     is >> c; // [
114
```

```
115     for (int y=0; y<alto; y++) {
116       Pixel1DContainer fila;
117       for (int x=0; x<ancho; x++) {
118         Pixel p;
119         p.cargar(is);
120         fila.push_back(p);
121         is >> c; // ] o ,
122       }
123       this->pixels.push_back(fila);
124     }
125
126 }
127
128 //auxiliares para posicionesMasOscuras
129
130 int Imagen::sumaCanales(int x, int y) const {
131     Pixel p = this->pixels[y][x];
132     return p.red() + p.green() + p.blue();
133 }
134
135 int Imagen::colorMasOscuro() const {
136     int x=0; int y=0;
137     int colorMasOscuro = this->sumaCanales(0,0);
138     while (x < this->ancho()){
139     y=0;
140         while (y < this->alto()){
141             if (colorMasOscuro > this->sumaCanales(x,y)){
142                 colorMasOscuro = this->sumaCanales(x,y);
143             }
144             y++;
145         }
146         x++;
147     }
148     return colorMasOscuro;
149 }
150
151 // auxiliares para blur y acuarela
152
153 int Imagen::maxPos(const int a[], int desde, int hasta) const{
154     int maxPos = desde;
155     int i = desde + 1;
156     while (i <= hasta) {
157         if (a[i] > a[maxPos]) maxPos = i;
158         i++;
159     }
160     return maxPos;
161 }
162
163 void Imagen::upSort(int a[], int n) const{
164     int pos; int valorActual; int actual = n - 1;
165     while (actual > 0) {
166         pos = maxPos(a ,0 ,actual);
167         valorActual = a[actual];
168         a[actual] = a[pos];
169         a[pos] = valorActual;
170         actual--;
171     }
172 }
```

```
173
174 bool Imagen::kVecinosCompletos(int k, int x, int y) const {
175
176     return (x-k+1)>=0
177         && (x+k-1)<this->ancho()
178         && (y-k+1)>=0
179         && (y+k-1)<this->alto();
180
181 }
182
183 void Imagen::aplicarFiltro(int filtro, int k) {
184     Pixel2DContainer pixelsFiltro;
185     int x=0; int y=0;
186     Pixel pixel(0,0,0);
187
188     while (y < this->alto()){
189         Pixel1DContainer fila;
190         while(x < this->ancho()){
191             if(this->kVecinosCompletos(k,x,y)){
192                 switch(filtro){
193                     case BLUR:
194                         pixel = this->pixelPromedioKVecinos(k,x,y);
195                         break;
196                     case ACUARELA:
197                         pixel = this->pixelMedianaKVecinos(k,x,y);
198                         break;
199                 }
200             }
201             fila.push_back(pixel);
202             pixel.cambiarPixel(0,0,0);
203             x++;
204         }
205         pixelsFiltro.push_back(fila);
206         x=0;
207         y++;
208     }
209
210     this->pixels.clear();
211     this->pixels = pixelsFiltro;
212 }
213
214 Pixel Imagen::pixelPromedioKVecinos(int k, int x, int y) const {
215     int r=0; int g=0; int b=0;
216     int totalKVecinos=(2*k-1)*(2*k-1);
217     int xi=x-k+1;
218     int yi=y-k+1;
219
220     Pixel pixelPromedio;
221
222     while(yi < y+k){
223         Pixel p;
224         while(xi < x+k){
225             p = this->obtenerPixel(yi,xi);
226             r += p.red();
227             g += p.green();
228             b += p.blue();
229             xi++;
230         }
```

```
231         xi=x-k+1;
232         yi++;
233     }
234
235     r /= totalKVecinos;
236     g /= totalKVecinos;
237     b /= totalKVecinos;
238
239     pixelPromedio.cambiarPixel(r,g,b);
240
241     return pixelPromedio;
242 }
243
244 Pixel Imagen::pixelMedianaKVecinos(int k, int x, int y) const {
245   int xi = x-k+1;
246   int yi = y-k+1;
247   int sizeKVecinos = (2*k-1)*(2*k-1);
248   int middleKVecinos = sizeKVecinos/2;
249   int* red = new int[sizeKVecinos];
250   int* green = new int[sizeKVecinos];
251   int* blue = new int[sizeKVecinos];
252   int i = 0;
253
254   Pixel pixelMediana;
255
256   while(yi < y+k){
257     Pixel p;
258     while(xi < x+k){
259       p = this->obtenerPixel(yi,xi);
260       red[i] = p.red();
261       green[i] = p.green();
262       blue[i] = p.blue();
263
264       i++;
265       xi++;
266     }
267     xi=x-k+1;
268     yi++;
269   }
270
271   upSort(red, sizeKVecinos);
272   upSort(green, sizeKVecinos);
273   upSort(blue, sizeKVecinos);
274
275   pixelMediana.cambiarPixel(red[middleKVecinos],
276                             green[middleKVecinos],
277                             blue[middleKVecinos]);
278   return pixelMediana;
279 }
```

3.5. galeria$_i$magenes.h

```
#ifndef GALERIA_IMAGENES_H
#define GALERIA_IMAGENES_H

#include "imagen.h"

#include <vector>

class GaleriaImagenes {
 public:
  void dividirYAgregar(const Imagen &imagen, int n, int m);
  Imagen laMasChiquitaConPuntoBlanco() const;
  void agregarImagen(const Imagen &imagen);
  void votar(const Imagen &imagen);
  void eliminarMasVotada();
  vector <Imagen> top10() const;

  void guardar(std::ostream& os) const;
  void cargar (std::istream& is);

 private:
  void acomodar();
  bool existeImagen(const Imagen &imagen);
  std::vector<Imagen> imagenes;
  std::vector<int> votos;
};

#endif  // GALERIA_IMAGENES_H
```

3.6. galeria_imagenes.cpp

```cpp
#include "imagen.h"
#include "galeria_imagenes.h"
#include <vector>

void GaleriaImagenes::dividirYAgregar(const Imagen &imagen, int n, int m) {
  int d, i, j, x, y, k, an, al;
  vector<Imagen> divisiones;

  if ( this->existeImagen(imagen)
    && (n > 0)
    && (m > 0)
    && (n % imagen.ancho() == 0)
    && (m % imagen.alto() == 0)
  ) {

    k = n*m;
    an = imagen.ancho() / n;
    al = imagen.alto() / m;

    for(d=0; d<k; d++) {
      divisiones.push_back(Imagen(an, al));
    }

    for(x=0; x<n; x++) {
      for(y=0; y<m; y++) {
        d = x*m + y;

        for (j=0; j<an; j++) {
          for (i=0; i<al; i++) {
            Pixel p = imagen.obtenerPixel(x*an + j, y*al + i);
            divisiones[d].modificarPixel(j, i, p);
          }
        }

      }
    }

    for(d=0; d<k; d++) {
      this->agregarImagen(divisiones[d]);
    }

  }
}

Imagen GaleriaImagenes::laMasChiquitaConPuntoBlanco() const {
  int x; int y;
  int an; int al;
  int area;
  bool imgTieneBlanco = false;

  int minArea = -1;
  Pixel pixelBlanco (255, 255, 255);
  Imagen laMasChiquitaConPuntoBlanco (0, 0);
  Imagen img (0, 0);

  // recorro todas las imágenes buscando si tienen al menos un punto blanco
```

```
57    for(int i=0; i < this->imagenes.size(); i++) {
58      img = this->imagenes[i];
59      imgTieneBlanco = false;
60
61      an = img.ancho();
62      al = img.alto();
63      area = an * al;
64
65      y = 0;
66      while(y < al && !imgTieneBlanco) {
67        x=0;
68        while(x < an && !imgTieneBlanco) {
69
70          // en el caso de que tengan punto blanco me fijo si es más chica la
71          // imagen que la anterior que tuviese
72          if(img.obtenerPixel(y, x) == pixelBlanco) {
73            imgTieneBlanco = true;
74            // si todavia no habia guardado ninguna con punto blanco o el area
75            // que tengo es mas grande que esta, la guardo
76            if ((minArea == -1) || (minArea > area)) {
77                minArea = area;
78                laMasChiquitaConPuntoBlanco = img;
79            }
80          }
81
82          x++;
83        }
84
85        y++;
86      }
87
88    }
89
90    return laMasChiquitaConPuntoBlanco;
91  }
92
93  void GaleriaImagenes::agregarImagen(const Imagen &imagen) {
94
95    if(!this->existeImagen(imagen)){
96      this->imagenes.push_back(imagen);
97      this->votos.push_back(0);
98      this->acomodar();
99    }
100 }
101
102 void GaleriaImagenes::votar(const Imagen &imagen) {
103   int posImg;
104   int k=0;
105   while(k<this->imagenes.size()){
106     if(this->imagenes[k]==imagen){
107       posImg=k;
108     }
109     k++;
110   }
111   this->votos[posImg]++;
112   this->acomodar();
113 }
114
```

```
115 void GaleriaImagenes::eliminarMasVotada() {
116   this->imagenes.pop_back();
117   this->votos.pop_back();
118 }
119
120 vector <Imagen> GaleriaImagenes::top10() const {
121     int k = this->imagenes.size() - 1;
122     int i;
123     vector <Imagen> res;
124
125     i = k;
126     while ((i > k-10) && (i >= 0)) {
127         res.push_back(this->imagenes[i]);
128         i--;
129     }
130
131     return res;
132 }
133
134 void GaleriaImagenes::guardar (std::ostream& os) const {
135
136   int i = 0;
137   os << '[';
138
139   while(i < this->imagenes.size()) {
140
141       if (i>0) {
142         os << ',';
143       }
144
145       os << '(';
146       this->imagenes[i].guardar(os);
147       os << ',';
148       os << this->votos[i];
149       os << ')';
150       i++;
151   }
152
153   os << ']';
154
155 }
156
157 void GaleriaImagenes::cargar (std::istream& is) {
158
159   Imagen img(0, 0);
160   int vts;
161   char c;
162
163   this->imagenes.clear();
164   this->votos.clear();
165
166   while(is >> c) { // [,]
167     if (is >> c) { // (
168
169       img.cargar(is);
170       is >> c; // ,
171       is >> vts;
172       is >> c; // )
```

```
173         this->imagenes.push_back(img);
174         this->votos.push_back(vts);
175       }
176     }
177     this->acomodar();
178   }
179
180   void GaleriaImagenes::acomodar() {
181     Imagen ai(0,0);
182     int    av;
183
184     int i, j;
185
186     for(i=0; i<this->votos.size(); i++) {
187       for(j=i+1; j<this->votos.size(); j++){
188
189         if (this->votos[i] > this->votos[j]) {
190           ai = this->imagenes[i];
191           av = this->votos[i];
192
193           this->imagenes[i] = this->imagenes[j];
194           this->votos[i] = this->votos[j];
195
196           this->imagenes[j] = ai;
197           this->votos[j] = av;
198         }
199       }
200     }
201   }
202
203   bool GaleriaImagenes::existeImagen(const Imagen &imagen) {
204     int i=0;
205     bool res=false;
206     while(i<this->imagenes.size() && res==false){
207       if(imagen == this->imagenes[i]){
208         res=true;
209       }
210       i++;
211     }
212     return res;
213   }
```

3.7. main.cpp

```
#include "pixel.h"
#include "imagen.h"
#include "galeria_imagenes.h"

#include <iostream>
#include <fstream>

using namespace std;

void doFilter(int filter);
void doCargarGaleria(GaleriaImagenes &galeria, string &rutaArchivoGaleria);
void doDividirYAgregar(GaleriaImagenes &galeria);
void doPosicionesMasOscuras(GaleriaImagenes &galeria);
void doTop10(GaleriaImagenes &galeria);
void doLaMasChiquitaConPuntoBlanco(GaleriaImagenes &galeria);
void doAgregarImagen(GaleriaImagenes &galeria);
void doVotar(GaleriaImagenes &galeria);
void doEliminarMasVotada(GaleriaImagenes &galeria);
void doGuardarGaleria(GaleriaImagenes &galeria, string &rutaArchivoGaleria);

void abrirArchivoIn(string mensaje, ifstream &archivoIn);
void abrirArchivoInYpisarElNombre(string msj, ifstream &archivoIn, string& nom);
void abrirArchivoOut(string mensaje, ofstream &archivoOut);


int main()
{
        int selected_action =- 1;
        GaleriaImagenes galeria;
        string rutaArchivoGaleria;

        while(selected_action != 0){

                cout << "ingrese el número de la acción que desea realizar:\n"
                " 0. exit\n"
                " 1. blur\n"
                " 2. acuarela\n"
                " 3. cargar galería\n"
                " 4. dividirYAgregar\n"
                " 5. posicionesMásOscuras\n"
                " 6. top 10\n"
                " 7. laMásChiquitaConPuntoBlanco\n"
                " 8. agregarImagen\n"
                " 9. votar\n"
                "10. eliminarMásVotada\n"
                "11. guardar galería\n"
                ;

                cin >> selected_action;

                switch (selected_action){
      case 1:  doFilter(BLUR);
               break;

      case 2:  doFilter(ACUARELA);
               break;
```

```
57
58          case 3:  doCargarGaleria(galeria, rutaArchivoGaleria);
59                   break;
60
61          case 4:  doDividirYAgregar(galeria);
62                   break;
63
64          case 5:  doPosicionesMasOscuras(galeria);
65                   break;
66
67          case 6:  doTop10(galeria);
68                   break;
69
70          case 7:  doLaMasChiquitaConPuntoBlanco(galeria);
71                   break;
72
73          case 8:  doAgregarImagen(galeria);
74                   break;
75
76          case 9:  doVotar(galeria);
77                   break;
78
79          case 10: doEliminarMasVotada(galeria);
80                   break;
81
82          case 11: doGuardarGaleria(galeria, rutaArchivoGaleria);
83                   break;
84
85                   }
86
87                   if(selected_action != 0){
88                           cin.ignore();
89                           cout << "presione una tecla para continuar...\n";
90                           cin.ignore();
91                   }
92          }
93  }
94
95  void doFilter(int filter){
96          Imagen imagenOriginal (0,0);
97          ifstream archivoIn;
98          abrirArchivoIn("ingrese el nombre del archivo a modificar:", archivoIn);
99
100         ofstream archivoOut;
101         abrirArchivoOut("ingrese el nombre del archivo de salida:", archivoOut);
102
103         int k;
104
105         cout << "ingrese la intensidad del filtro (k):\n";
106         cin >> k;
107
108         imagenOriginal.cargar(archivoIn);
109         switch(filter){
110                 case BLUR:
111                         imagenOriginal.blur(k);
112                         break;
113                 case ACUARELA:
114                         imagenOriginal.acuarela(k);
```

```
115                         break;
116         }
117
118         imagenOriginal.guardar(archivoOut);
119 }
120
121 void doCargarGaleria(GaleriaImagenes &galeria, string &rutaArchivoGaleria){
122         ifstream archivoIn;
123
124         abrirArchivoInYpisarElNombre(
125                 "ingrese el nombre de la galeria que desea cargar:",
126                 archivoIn, rutaArchivoGaleria);
127
128         galeria.cargar(archivoIn);
129 }
130
131 void doDividirYAgregar(GaleriaImagenes &galeria){
132         int n, m;
133         Imagen im (0,0);
134         ifstream archivoIn;
135         abrirArchivoIn("Nombre del archivo de la imagen (que está en la galeria)"
136                                 " a agregar sus divisiones?", archivoIn);
137         im.cargar(archivoIn);
138
139         cout << "Cuantas columnas? (n, debe ser multiplo del ancho de la img):\n";
140          cin >> n;
141
142         cout << "Cuantas filas? (m, debe ser multiplo del alto de la img):\n";
143         cin >> m;
144
145         galeria.dividirYAgregar(im, n, m);
146 }
147
148 void doPosicionesMasOscuras(GaleriaImagenes &galeria){
149         string nombreArchivoIn;
150         ifstream archivoIn;
151         abrirArchivoIn("ingrese el nombre del archivo a leer:", archivoIn);
152
153         Imagen imagenOriginal (0,0);
154
155         imagenOriginal.cargar(archivoIn);
156
157         vector < pair <int, int> > pos;
158         pos = imagenOriginal.posicionesMasOscuras();
159
160         int i;
161
162         cout << "[";
163
164         for(i=0; i<pos.size() ; i++){
165                 if(i == pos.size()-1){
166                         cout << "(" << pos[i].first << "," << pos[i].second << ")";
167                 }else{
168                         cout << "(" << pos[i].first << "," << pos[i].second << "),";
169                 }
170         }
171
172         cout << "]\n";
```

```
173
174         galeria.agregarImagen(imagenOriginal);
175
176 }
177
178 void doTop10(GaleriaImagenes &galeria){
179         vector<Imagen> top10;
180         ofstream archivoOut;
181         abrirArchivoOut("ingrese el nombre del archivo de salida:", archivoOut);
182
183         top10 = galeria.top10();
184
185         int i = 0;
186
187         archivoOut << '[';
188
189         while(i < top10.size()) {
190                         if (i>0 && i < top10.size()) {
191                         archivoOut << ',';
192                         }
193                         top10[i].guardar(archivoOut);
194                         i++;
195         }
196
197         archivoOut << ']';
198 }
199
200 void doLaMasChiquitaConPuntoBlanco(GaleriaImagenes &galeria){
201         Imagen imagenFinal = galeria.laMasChiquitaConPuntoBlanco();
202
203         if(imagenFinal.alto() != 0){
204                 ofstream archivoOut;
205                 abrirArchivoOut("ingrese el nombre para guardar la imagen más chiquita"
206                                                 " con punto blanco:", archivoOut);
207
208                 imagenFinal.guardar(archivoOut);
209         }
210         else{
211                 cout << "no se encontró ninguna imagen con punto blanco\n";
212         }
213 }
214
215 void doAgregarImagen(GaleriaImagenes &galeria){
216         ifstream archivoIn;
217         abrirArchivoIn("ingrese el nombre de "
218                                         "la imagen que desea agregar:", archivoIn);
219
220         Imagen imagenNueva (0,0);
221
222         imagenNueva.cargar(archivoIn);
223
224         galeria.agregarImagen(imagenNueva);
225
226 }
227
228 void doVotar(GaleriaImagenes &galeria){
229         Imagen imagenVotada (0,0);
230
```

```
231         ifstream archivoIn;
232         abrirArchivoIn("ingrese el nombre de la imagen a votar:", archivoIn);
233 
234         imagenVotada.cargar(archivoIn);
235 
236         galeria.votar(imagenVotada);
237 }
238 
239 void doEliminarMasVotada(GaleriaImagenes &galeria){
240         galeria.eliminarMasVotada();
241 }
242 
243 void doGuardarGaleria(GaleriaImagenes &galeria, string &rutaArchivoGaleria){
244 
245         ofstream archivoOut;
246         archivoOut.open(rutaArchivoGaleria.c_str());
247 
248         galeria.guardar(archivoOut);
249 }
250 
251 string pedirString(string mensaje) {
252         string str;
253 
254         cout << mensaje << '\n';
255         cin  >> str;
256 
257         return str;
258 }
259 
260 void abrirArchivoIn(string mensaje, ifstream& archivoIn) {
261         archivoIn.open(pedirString(mensaje).c_str());
262 }
263 
264 void abrirArchivoInYpisarElNombre(string mje, ifstream& archivoIn, string& nom){
265         nom = pedirString(mje);
266         archivoIn.open(nom.c_str());
267 }
268 
269 void abrirArchivoOut(string mensaje, ofstream& archivoOut) {
270         archivoOut.open(pedirString(mensaje).c_str());
271 }
```

3.8. tests.cpp

```cpp
#include "pixel.h"
#include "imagen.h"
#include "galeria_imagenes.h"

#include <iostream>
#include <fstream>
#include <sstream>
#include <cassert>

using namespace std;

void testPixel();
void testImagen();
void testGaleria();
int main()
{
    testPixel();
    testImagen();
    testGaleria();
    cout << "Tests OK" << endl;
}

void testPixel() {

  // constructor con argumentos
  Pixel pc1 (4, 4, 4);
  assert (pc1.red()    == 4);
  assert (pc1.green()  == 4);
  assert (pc1.blue()   == 4);

  // constructor con argumentos fuera de rango va a guardar fruta
  Pixel pcfr (4, 4, 46666);
  assert (pcfr.red()    == 4);
  assert (pcfr.green()  == 4);
  assert (pcfr.blue()   == 46666);

  // igualdad
  Pixel p1, p2;
  assert (p1 == p2);

  // cambiarPixel
  p1.cambiarPixel(12, 34, 56);
  assert (p1.red()    == 12);
  assert (p1.green()  == 34);
  assert (p1.blue()   == 56);

  // desigualdad
  assert (p1 != p2);

  //guardar
  Pixel g(10,50,3);
  ostringstream oss;
  g.guardar(oss);
  assert (oss.str() == "(10;50;3)");

  //cargar
```

```
57    Pixel c;
58    istringstream iss ("(60;50;40)");
59    c.cargar(iss);
60    assert (c.red()   == 60);
61    assert (c.green() == 50);
62    assert (c.blue()  == 40);
63  }
64
65  void testImagen() {
66    // constructor, alto, ancho, todanegra
67    Pixel negro;
68    Pixel rojo(255, 0, 0);
69    Pixel p;
70    Imagen i1 (2, 3);
71    assert(i1.alto() == 2);
72    assert(i1.ancho() == 3);
73
74    for (int y=0; y < i1.alto(); y++) {
75      for (int x=0; x < i1.ancho(); x++) {
76          assert(i1.obtenerPixel(y, x) == negro);
77      }
78    }
79
80    assert(i1.obtenerPixel(1,1) == negro);
81    i1.modificarPixel(1,1, rojo);
82    assert(i1.obtenerPixel(1,1) == rojo);
83
84    //guardar
85    ostringstream oss;
86    i1.guardar(oss);
87    string esperado = "2 3 [(0;0;0),(0;0;0),(0;0;0),(0;0;0),(255;0;0),(0;0;0)]";
88    assert (oss.str() == esperado);
89
90    //cargar
91    istringstream iss("2 2 [(1;2;3),(123;121;31),(2;1;231),(167;161;173)]");
92    Imagen ic(0,0);
93    ic.cargar(iss);
94    assert (ic.obtenerPixel(0,0) == Pixel(1,    2,  3));
95    assert (ic.obtenerPixel(0,1) == Pixel(123,121, 31));
96    assert (ic.obtenerPixel(1,0) == Pixel(2,    1,231));
97    assert (ic.obtenerPixel(1,1) == Pixel(167,161,173));
98
99    //blur
100   ifstream archivoOriginal("../res/hermione.specimg");
101   ifstream archivoOriginal2("../res/hermione.specimg");
102
103   ifstream archivoBlurK5("../res/hermione.blur.k5.specimg");
104   ifstream archivoAcuarelaK5("../res/hermione.acuarela.k5.specimg");
105
106   Imagen imagenOriginal(0,0);
107   Imagen imagenOriginal2(0,0);
108
109   Imagen imagenBlurK5(0,0);
110   Imagen imagenAcuarelaK5(0,0);
111
112   imagenOriginal.cargar(archivoOriginal);
113   imagenOriginal2.cargar(archivoOriginal2);
114
```

```
115   imagenOriginal.blur(5);
116   imagenOriginal2.acuarela(5);
117
118   imagenBlurK5.cargar(archivoBlurK5);
119   imagenAcuarelaK5.cargar(archivoAcuarelaK5);
120
121   assert (imagenOriginal == imagenBlurK5);
122   assert (imagenOriginal2 == imagenAcuarelaK5);
123
124   //posicionesMasOscuras
125   //ejemplo fruta
126   Imagen ipmo(2,2);
127   ipmo.modificarPixel(0,0, rojo);
128   ipmo.modificarPixel(1,1, rojo);
129
130   vector < pair <int, int> > pos;
131   pos = ipmo.posicionesMasOscuras();
132
133   assert(pos[0] == (pair <int, int>(0, 1)));
134   assert(pos[1] == (pair <int, int>(1, 0)));
135
136   //poder calcularlo en imagenes de verdad
137   vector < pair <int, int> > hermioneBlurPosicionesMasOscuras;
138   hermioneBlurPosicionesMasOscuras = imagenBlurK5.posicionesMasOscuras();
139
140
141 }
142
143 void testGaleria() {
144   string galeriaInString  =
145                 "[(1 1 [(1;1;1)],5),(1 1 [(3;3;3)],3),(1 1 [(2;2;2)],2)]";
146   string galeriaOutString =
147                 "[(1 1 [(2;2;2)],2),(1 1 [(3;3;3)],3),(1 1 [(1;1;1)],5)]";
148
149   istringstream iss(galeriaInString.c_str());
150   ostringstream oss;
151
152   vector <Imagen> vim;
153   GaleriaImagenes g;
154   int i;
155   g.cargar(iss);
156   g.guardar(oss);
157   assert (oss.str() == galeriaOutString);
158
159   int res_t10_4[] = {4,3,2,1};
160   string gist10_4 = "["
161                         "(1 1 [(1;1;1)],1),"
162                         "(1 1 [(4;4;4)],4),"
163                         "(1 1 [(3;3;3)],3),"
164                         "(1 1 [(2;2;2)],2)"
165                     "]";
166   int res_t10_10[] = {9,8,7,6,5,4,3,2,1,0};
167   string gist10_10 = "["
168                         "(1 1 [(1;1;1)],1),"
169                         "(1 1 [(4;4;4)],4),"
170                         "(1 1 [(3;3;3)],3),"
171                         "(1 1 [(2;2;2)],2),"
172                         "(1 1 [(6;6;6)],6),"
```

```
173                         "(1 1 [(8;8;8)],8),"
174                         "(1 1 [(9;9;9)],9),"
175                         "(1 1 [(5;5;5)],5),"
176                         "(1 1 [(7;7;7)],7),"
177                         "(1 1 [(0;0;0)],0)"
178                     "]";
179     int res_t10_12[] = {500,100,9,8,7,6,5,4,3,2};
180     string gist10_12 = "["
181                         "(1 1 [(1;1;1)],1),"
182                         "(1 1 [(4;4;4)],4),"
183                         "(1 1 [(3;3;3)],3),"
184                         "(1 1 [(2;2;2)],2),"
185                         "(1 1 [(6;6;6)],6),"
186                         "(1 1 [(8;8;8)],8),"
187                         "(1 1 [(9;9;9)],9),"
188                         "(1 1 [(5;5;5)],5),"
189                         "(1 1 [(7;7;7)],7),"
190                         "(1 1 [(0;0;0)],0),"
191                         "(1 1 [(100;100;100)],100),"
192                         "(1 1 [(500;500;500)],500)"
193                       "]";
194
195     istringstream iss4(gist10_4.c_str());
196     g.cargar(iss4);
197
198     vim = g.top10();
199     for (i=0; i<vim.size(); i++) {
200       assert(vim[i].obtenerPixel(0,0).red() == res_t10_4[i]);
201     }
202
203     istringstream iss10(gist10_10.c_str());
204     g.cargar(iss10);
205
206     vim = g.top10();
207     for (i=0; i<vim.size(); i++) {
208       assert(vim[i].obtenerPixel(0,0).red() == res_t10_10[i]);
209     }
210
211     istringstream iss12(gist10_12.c_str());
212     g.cargar(iss12);
213
214     vim = g.top10();
215     for (i=0; i<vim.size(); i++) {
216       assert(vim[i].obtenerPixel(0,0).red() == res_t10_12[i]);
217     }
218
219     //agrega imagen cuaquiera de 2x2 que no está
220     string galeriaInStringa  = "[(1 1 [(1;1;1)],5),(1 1 [(3;3;3)],3),"
221                                "(1 1 [(2;2;2)],2)]";
222
223     string galeriaOutStringa =
224                       "[(2 2 [(1;2;3),(123;121;31),(2;1;231),(167;161;173)],0),"
225                       "(1 1 [(2;2;2)],2),(1 1 [(3;3;3)],3),(1 1 [(1;1;1)],5)]";
226
227     istringstream issa(galeriaInStringa.c_str());
228     ostringstream ossa;
229
230     GaleriaImagenes ga;
```

```
231     ga.cargar(issa);
232
233     istringstream ihss("2 2 [(1;2;3),(123;121;31),(2;1;231),(167;161;173)]");
234
235     Imagen nueva(0,0);
236     nueva.cargar(ihss);
237
238     ga.agregarImagen(nueva);
239     ga.guardar(ossa);
240     assert (ossa.str() == galeriaOutStringa);
241
242     //mantiene la galeria porque la imagen y existia
243
244     string galeriaInStringb  =
245               "[(1 1 [(2;2;2)],2),(1 1 [(3;3;3)],3),(1 1 [(1;1;1)],5)]";
246     string galeriaOutStringb =
247               "[(1 1 [(2;2;2)],2),(1 1 [(3;3;3)],3),(1 1 [(1;1;1)],5)]";
248     istringstream issb(galeriaInStringb.c_str());
249     ostringstream ossb;
250
251
252     GaleriaImagenes gb;
253     gb.cargar(issb);
254
255     istringstream inss("1 1 [(2;2;2)]");
256
257     Imagen usada(0,0);
258     usada.cargar(inss);
259
260     gb.agregarImagen(usada);
261     gb.guardar(ossb);
262     assert (ossb.str() == galeriaOutStringb);
263
264     //galeria vacia
265
266     string galeriaInStringc  = "";
267     string galeriaOutStringc = "[(1 1 [(2;2;2)],0)]";
268     istringstream issc(galeriaInStringc.c_str());
269     ostringstream ossc;
270
271
272     GaleriaImagenes gc;
273     gc.cargar(issc);
274
275     istringstream ijss("1 1 [(2;2;2)]");
276
277     Imagen rara(0,0);
278     rara.cargar(ijss);
279
280     gc.agregarImagen(rara);
281     gc.guardar(ossc);
282     assert (ossc.str() == galeriaOutStringc);
283
284     // la más chiquita con punto blanco
285
286     // una imagen con punto blanco
287     Imagen laMasChiquita(0,0);
288     galeriaInString = "[(1 1 [(0;0;0)],0),(1 1 [(255;255;255)],0)]";
```

```
289   galeriaOutString = "1 1 [(255;255;255)]";
290   iss.clear();
291   iss.str(galeriaInString.c_str());
292
293   g.cargar(iss);
294
295   laMasChiquita = g.laMasChiquitaConPuntoBlanco();
296
297   oss.str("");
298   oss.clear();
299   laMasChiquita.guardar(oss);
300   assert (oss.str() == galeriaOutString);
301
302   // dos imagenes con punto blanco
303   galeriaInString =
304         "[(1 2 [(255;255;255),(0;0;0)],0),(1 1 [(0;0;0)],0),"
305         "(3 1 [(255;255;255),(0;0;0),(255;255;255)],0)]";
306
307   galeriaOutString = "1 2 [(255;255;255),(0;0;0)]";
308   iss.clear();
309   iss.str(galeriaInString.c_str());
310
311   g.cargar(iss);
312
313   laMasChiquita = g.laMasChiquitaConPuntoBlanco();
314
315   oss.str("");
316   oss.clear();
317   laMasChiquita.guardar(oss);
318   assert (oss.str() == galeriaOutString);
319
320
321
322   //votar imagen
323   g.votar(laMasChiquita);
324   oss.str("");
325   oss.clear();
326   g.guardar(oss);
327   assert (oss.str() ==
328     "[(1 1 [(0;0;0)],0),(3 1 [(255;255;255),(0;0;0),(255;255;255)],0),"
329     "(1 2 [(255;255;255),(0;0;0)],1)]");
330
331   //dividir y agregar
332   {
333     istringstream giss(
334                 "[(2 2 [(11;11;11),(22;22;22),(33;33;33),(44;44;44)],0)]");
335
336     istringstream iiss("2 2 [(11;11;11),(22;22;22),(33;33;33),(44;44;44)]");
337     ostringstream oss;
338     GaleriaImagenes g;
339     Imagen im(0,0);
340
341     g.cargar(giss);
342     im.cargar(iiss);
343
344     g.dividirYAgregar(im, 2, 2);
345
346     g.guardar(oss);
```

```
347
348     assert (oss.str() ==
349         "[(2 2 [(11;11;11),(22;22;22),(33;33;33),(44;44;44)],0),"
350         "(1 1 [(11;11;11)],0),(1 1 [(22;22;22)],0),(1 1 [(33;33;33)],0),"
351         "(1 1 [(44;44;44)],0)]");
352   }
353 }
```

3.9. Makefile

```
all: clean tests main

clean:
rm -f pixel.o imagen.o galeria_imagenes.o main tests

tests: pixel.o imagen.o galeria_imagenes.o conv
g++ -g -o tests tests.cpp pixel.o imagen.o galeria_imagenes.o
./tests

main: pixel.o imagen.o galeria_imagenes.o
g++ -g -o main main.cpp pixel.o imagen.o galeria_imagenes.o

pixel.o:
g++ -g -c pixel.cpp

imagen.o:
g++ -g -c imagen.cpp

galeria_imagenes.o:
g++ -g -c galeria_imagenes.cpp
```