



Trabajo Práctico Número 2

Algoritmos y Estructuras de Datos I

Grupo: 1

Integrante	LU	Correo electrónico
Ciruelos Rodríguez, Gonzalo	063/14	gonzalo.ciruelos@gmail.com
Gatti, Mathias	477/14	mathigatti@gmail.com
Rabinowicz, Lucía	105/14	lu.rabinowicz@gmail.com
Weber, Andres	923/13	herr.andyweber@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. Observaciones

1. Dado que en el ejercicio de la demostracion nos era dificultoso implicar los aseguras del problema, los docentes nos sugirieron cambiar la especificacion de pop_back por una mas fuerte, que es la que sigue:

```

problema pop_back ( $a : [T]$ ) {
  requiere :  $|a| > 0$ ;
  modifica:  $a$ ;
  asegura sacaElUltimo:  $pre(a) == a + [pre(a)[|pre(a)| - 1]]$ ;
}

```

2. Especificación

2.1. posicionesMasOscuras

```

problema posicionesMasOscuras ( $im : Imagen$ ) =  $result : [\mathbb{Z}, \mathbb{Z}]$  {
  asegura :  $mismos(result, [(i, j) | i \leftarrow [0..ancho(im)], j \leftarrow [0..alto(im)], esMinimo(sumaColor(color(im, i, j)), im)])$ ;
  aux sumaColor( $p : Pixel$ ): $\mathbb{Z} = red(p) + green(p) + blue(p)$ ;
  aux esMinimo( $s : \mathbb{Z}, im : Imagen$ ) :  $Bool = (\forall i \leftarrow [0..ancho(im)], \forall j \leftarrow [0..alto(im)]) s \leq sumaColor(color(im, i, j))$ ;
}

```

2.2. top10

```

problema top10 ( $g : Galeria$ ) =  $result : [Imagen]$ 
  asegura :  $0 \leq |result| \leq 10$ ;
  asegura :  $|imagenes(g)| \leq 10 \implies mismos(result, imagenes(g))$ ;
  asegura :  $|imagenes(g)| > 10 \implies |result| == 10$ ;
  asegura :  $(\forall h \leftarrow result) h \in imagenes(g)$ ;
  asegura :  $|imagenes(g)| > 10 \implies [votos(g, i) | i \leftarrow result] == ordenarDecr([votos(g, i) | i \leftarrow imagenes(g)])[0..10]$ ;
  asegura :  $ordenadaDecreciente([votos(g, result_i) | i \leftarrow [0..|result|]])$ ;
  aux ordenadaDecreciente( $l : [\mathbb{Z}]$ ) :  $Bool = (\forall i, j \leftarrow [0..|l|], i \geq j) l_j \geq l_i$ ;
  aux ordenarDecr( $l : [\mathbb{Z}]$ ) :  $[\mathbb{Z}] = [x \mid n \leftarrow [0..|l|], x \leftarrow l, cantidadMenores(l, x) == n]$ ;
  aux cantidadMenores( $l : [\mathbb{Z}], x : \mathbb{Z}$ ) :  $\mathbb{Z} = |[y \mid y \leftarrow l, y < x]|$ ;
}

```

2.3. laMasChiquitaConPuntoBlanco

```

problema laMasChiquitaConPuntoBlanco ( $g : Galeria$ ) =  $result : Imagen$  {
  requiere existeImagenConPuntoBlanco:  $(\exists h \leftarrow imagenes(g)) tieneBlanco(h)$ ;
  asegura :  $tieneBlanco(result)$ ;
  asegura esLaMasChica:  $(\forall j \leftarrow imagenesConBlanco(g)) ancho(result) * alto(result) \leq ancho(j) * alto(j)$ ;
  aux imagenesConBlanco ( $g : Galeria$ ) :  $[Imagen] = [h \mid h \leftarrow imagenes(g), tieneBlanco(h)]$ ;
  aux tieneBlanco( $i : Imagen$ ) :  $Bool = (\exists x \leftarrow [0..ancho(i)], y \leftarrow [0..alto(i)]) sumaColor(color(i, x, y)) == 255 * 3$ ;
}

```

```

    aux sumaColor( $p : Pixel$ ) :  $\mathbb{Z} = red(p) + green(p) + blue(p);$ 
}

```

2.4. agregarImagen

```

problema agregarImagen ( $g : Galeria, i : Imagen$ ) {
    requiere :  $i \notin imagenes(g)$ ;
    modifica:  $g$ ;
    asegura agregaImagen:  $mismos(imagenes(g), imagenes(pre(g)) + +[i])$ ;
    asegura :  $votos(g, i) == 0$ ;
    asegura noCambiaVotos:  $(\forall h \leftarrow imagenes(pre(g))) votos(g, h) == votos(pre(g), h)$ ;
}

```

2.5. votar

```

problema votar ( $g : Galeria, i : Imagen$ ) {
    requiere :  $i \in imagenes(g)$ ;
    modifica:  $g$ ;
    asegura noCambiaImagenes:  $mismos(imagenes(g), imagenes(pre(g)))$ ;
    asegura noCambiaVotosDelResto:  $(\forall h \leftarrow imagenes(g), h \neq i) votos(g, h) == votos(pre(g), h)$ ;
    asegura :  $votos(g, i) == votos(pre(g), i) + 1$ ;
}

```

2.6. eliminarMasVotada

```

problema eliminarMasVotada ( $g : Galeria$ ) {
    requiere :  $|imagenes(g)| > 0$ ;
    modifica:  $g$ ;
    asegura eliminaUnaImagen:  $[[h|h \leftarrow imagenes(pre(g)), h \notin imagenes(g)]] == 1$ ;
    asegura eliminaUnaYNoPoneNadaMas1:  $|imagenes(pre(g))| == |imagenes(g)| + 1$ ;
    asegura eliminaLaMasVotada:  $(\forall h \leftarrow imagenes(pre(g)))$ 
         $h \notin imagenes(g) \implies (\forall j \leftarrow imagenes(pre(g))) votos(pre(g), h) \geq votos(pre(g), j)$ ;
    asegura noCambiaVotos:  $(\forall h \leftarrow imagenes(g)) votos(pre(g), h) == votos(g, h)$ ;
}

```

¹Notese que estos dos asegura implican que $imagenes(g)$ es sublista de $imagenes(pre(g))$, dado que no tienen repetidos

3. Implementacion

Los headers (archivos .h) no fueron modificados con respecto a los originales, así que no los adjuntamos aquí.

Código fuente 1: pixel.cpp

```
1  #include "pixel.h"
2
3  Pixel::Pixel(int red, int green, int blue) {
4      intensidades[0] = red;
5      intensidades[1] = green;
6      intensidades[2] = blue;
7  }
8
9  void Pixel::cambiarPixel(int red, int green, int blue) {
10     intensidades[0] = red;
11     intensidades[1] = green;
12     intensidades[2] = blue;
13 }
14
15 int Pixel::red() const {
16     return intensidades[0];
17 }
18
19 int Pixel::green() const {
20     return intensidades[1];
21 }
22
23 int Pixel::blue() const {
24     return intensidades[2];
25 }
26
27 void Pixel::guardar(std::ostream& os) const {
28     os << "(" << intensidades[0] << ";" << intensidades[1] << ";" << intensidades[2] << ")";
29 }
30
31 void Pixel::cargar(std::istream& is) {
32     char charMolesto; //punto y coma o parentesis
33     is >> charMolesto;
34
35     int colores = 0;
36     while(colores < 3) {
37         int este_color;
38         is >> este_color;
39         intensidades[colores] = este_color;
40
41         is >> charMolesto;
42         colores++;
43     }
44 }
```

Código fuente 2: imagen.cpp

```
1  #include "imagen.h"
2
3  void sort(vector<int> &v) {
4      //algoritmo de burbujeo
5      int sz = v.size();
6      int i = 0;
7      while(i < sz) {
8          int j = 0;
9          while(j < sz - 1 - i) {
10             if (!(v[j] <= v[j + 1])) {
11                 int temp = v[j];
12                 v[j] = v[j + 1];
13                 v[j + 1] = temp;
14             }
15             j++;
16         }
17         i++;
18     }
19 }
20
```

```

21 Imagen::Imagen(int alto_param, int ancho_param) {
22     Pixel negro(0, 0, 0);
23     PixelIDContainer filaNegra(ancho_param, negro);
24
25     int i = 0;
26     while (i < alto_param) {
27         pixels.push_back(filaNegra);
28         i++;
29     }
30 }
31
32 Pixel Imagen::obtenerPixel(int fila, int columna) const {
33     return pixels[fila][columna];
34 }
35
36 void Imagen::modificarPixel(int fila, int columna, const Pixel &pixel) {
37     pixels[fila][columna] = pixel;
38 }
39
40 int Imagen::alto() const {
41     return pixels.size();
42 }
43
44 int Imagen::ancho() const {
45     return pixels[0].size();
46 }
47
48 int max(int a, int b) {
49     return a > b ? a : b;
50 }
51 int min(int a, int b) {
52     return a > b ? b : a;
53 }
54
55 vector<Pixel> kVecinos(Pixel2DContainer pixels, int pixel_i, int pixel_j, int k) {
56     int alto = pixels.size();
57     int ancho = pixels[0].size();
58
59     vector<Pixel> resultado;
60     int i = max(0, pixel_i - k + 1);
61     while(i < min(alto, pixel_i + k)) {
62         int j = max(0, pixel_j - k + 1);
63         while(j < min(ancho, pixel_j + k)) {
64             resultado.push_back(pixels[i][j]);
65             j++;
66         }
67         i++;
68     }
69
70     return resultado;
71 }
72
73 Pixel promedio(vector<Pixel> pixels){
74     int r = 0, g = 0, b = 0;
75     int cantidad = pixels.size();
76
77     int i = 0;
78     while (i < cantidad) {
79         r += pixels[i].red();
80         g += pixels[i].green();
81         b += pixels[i].blue();
82
83         i++;
84     }
85
86     Pixel resultado(r / cantidad, g / cantidad, b / cantidad);
87     return resultado;
88 }
89
90 void Imagen::blur(int k) {
91     int i = 0;
92
93     Pixel pixel_negro(0, 0, 0);

```

```

94
95     Pixel2DContainer nuevo_pixels;
96
97     Pixel1DContainer filaNegra(ancho(), pixel_negro);
98     int iter = 0;
99     while (iter < alto()) {
100         nuevo_pixels.push_back(filaNegra);
101         iter++;
102     }
103
104     int kVecCompletos = (2 * k - 1) * (2 * k - 1);
105     while (i < alto()) {
106         int j = 0;
107         while (j < ancho()) {
108             vector<Pixel> kVec = kVecinos(pixels, i, j, k);
109
110             if(kVec.size() == kVecCompletos)
111                 nuevo_pixels[i][j] = promedio(kVec);
112             else nuevo_pixels[i][j] = pixel_negro;
113
114             j++;
115         }
116
117         i++;
118     }
119
120     pixels = nuevo_pixels;
121 }
122
123 Pixel mediana(vector<Pixel> pixels) {
124     vector<int> reds, greens, blues;
125     int cantidad = pixels.size();
126
127     int i = 0;
128     while (i < cantidad) {
129         reds.push_back(pixels[i].red());
130         greens.push_back(pixels[i].green());
131         blues.push_back(pixels[i].blue());
132
133         i++;
134     }
135
136     sort(reds);
137     sort(greens);
138     sort(blues);
139
140     Pixel resultado(reds[cantidad / 2], greens[cantidad / 2], blues[cantidad / 2]);
141     return resultado;
142 }
143
144 void Imagen::acuarela(int k) {
145     Pixel pixel_negro(0, 0, 0);
146
147     Pixel2DContainer nuevo_pixels;
148
149     Pixel1DContainer filaNegra(ancho(), pixel_negro);
150     int iter = 0;
151     while (iter < alto()) {
152         nuevo_pixels.push_back(filaNegra);
153         iter++;
154     }
155
156     int i = 0;
157     while (i < alto()) {
158         int j = 0;
159         while (j < ancho()) {
160             vector<Pixel> kVec = kVecinos(pixels, i, j, k);
161
162             if(kVec.size() == (2 * k - 1) * (2 * k - 1))
163                 nuevo_pixels[i][j] = mediana(kVec);
164             else nuevo_pixels[i][j] = pixel_negro;
165
166             j++;

```

```

167         }
168         i++;
169     }
170     pixels = nuevo_pixels;
171 }
172
173
174
175 vector<pair<int, int> > Imagen::posicionesMasOscuras() const {
176     int colorMasOscuro = 255 * 3;
177
178     vector<pair<int, int> > resultado;
179
180     int i = 0;
181     while (i < alto()) {
182         int j = 0;
183         while (j < ancho()) {
184             Pixel estePixel = pixels[i][j];
185             int colorPixel = estePixel.red() + estePixel.green() + estePixel.blue();
186
187             if (colorPixel < colorMasOscuro) colorMasOscuro = colorPixel;
188
189             j++;
190         }
191         i++;
192     }
193
194
195     i = 0;
196     while (i < alto()) {
197         int j = 0;
198         while (j < ancho()) {
199             Pixel estePixel = pixels[i][j];
200             int colorPixel = estePixel.red() + estePixel.green() + estePixel.blue();
201
202             if (colorPixel == colorMasOscuro) resultado.push_back(make_pair(i, j));
203             j++;
204         }
205         i++;
206     }
207
208     return resultado;
209 }
210
211
212 bool Imagen::operator==(const Imagen &otra) const {
213     bool resultado = true;
214
215     if(alto() != otra.alto() || ancho() != otra.ancho())
216         resultado = false;
217     else {
218         int i = 0;
219         while (i < alto()) {
220             int j = 0;
221             while (j < ancho()) {
222                 Pixel p1 = pixels[i][j];
223                 Pixel p2 = otra.obtenerPixel(i, j);
224                 if (p1.red() != p2.red() ||
225                     p1.green() != p2.green() ||
226                     p1.blue() != p2.blue())
227                     resultado = false;
228                 j++;
229             }
230             i++;
231         }
232     }
233
234     return resultado;
235 }
236
237 void Imagen::guardar(std::ostream& os) const {
238     os << alto() << " " << ancho() << " ";
239     int i = 0;

```

```

240     os << "[";
241     while (i < alto()) {
242         int j = 0;
243         while (j < ancho()) {
244             if(i != 0 || j != 0) os << ",";
245             pixels[i][j].guardar(os);
246
247             j++;
248         }
249         i++;
250     }
251     os << "]\n";
252 }
253
254 void Imagen::cargar(std::istream& is) {
255     int alto, ancho;
256     is >> alto;
257     is >> ancho;
258
259     char charMolesto;
260     is >> charMolesto; //corchete o coma
261
262     vector<vector<Pixel> > nueva_imagen;
263
264     int i = 0;
265     while(i < alto) {
266         int j = 0;
267         vector<Pixel> fila;
268         while(j < ancho) {
269             Pixel este_pixel(0, 0, 0);
270             este_pixel.cargar(is);
271
272             is >> charMolesto;
273
274             fila.push_back(este_pixel);
275             j++;
276         }
277
278         nueva_imagen.push_back(fila);
279         i++;
280     }
281
282     pixels = nueva_imagen;
283 }

```

Código fuente 3: galeria_imagenes.cpp

```

1  #include "galeria_imagenes.h"
2
3
4  Imagen GaleriaImágenes::laMasChiquitaConPuntoBlanco () const {
5      vector<Imagen> imagenes_blancas;
6      vector<int> tamaño_imagenes;
7
8      int k = 0;
9      while (k < imagenes.size()) {
10         int n = imagenes[k].alto();
11         int m = imagenes[k].ancho();
12         int i = 0;
13         while (i < n) {
14             int j = 0;
15             while (j < m) {
16                 Pixel este_pixel = imagenes[k].obtenerPixel(i, j);
17                 if (este_pixel.red() == 255 && este_pixel.blue() == 255 && este_pixel.green() == 255) {
18                     imagenes_blancas.push_back(imagenes[k]);
19                     tamaño_imagenes.push_back(m * n);
20                     j = m; // para que termine el ciclo
21                     i = n; // porque ya encontramos el pixel
22                 }
23                 j++;
24             }
25             i++;
26         }

```



```
27     }
28     k++;
29 }
30
31 int mas_chica = tamaño_imagenes[0];
32 int i_mas_chica = 0;
33 int i = 1;
34 while(i < imagenes_blancas.size()) {
35     if(tamaño_imagenes[i] < mas_chica) {
36         i_mas_chica = i;
37         mas_chica = tamaño_imagenes[i];
38     }
39     i++;
40 }
41 return imagenes_blancas[i_mas_chica];
42
43 }
44
45 void GaleriaImagenes::agregarImagen(const Imagen &imagen) {
46     vector<Imagen> imagenes_nuevas;
47     vector<int> votos_nuevos;
48
49     imagenes_nuevas.push_back(imagen);
50     votos_nuevos.push_back(0);
51
52     int i = 0;
53     while(i < imagenes.size()) {
54         imagenes_nuevas.push_back(imagenes[i]);
55         votos_nuevos.push_back(votos[i]);
56
57         i++;
58     }
59     imagenes = imagenes_nuevas;
60     votos = votos_nuevos;
61 }
62
63 void GaleriaImagenes::votar(const Imagen &imagen) {
64     int i = 0;
65     std::vector<Imagen> imagenes_nuevas;
66     std::vector<int> votos_nuevos;
67
68     int votos_imagen;
69     while (i < imagenes.size() && !(imagenes[i] == imagen)) {
70         imagenes_nuevas.push_back(imagenes[i]);
71         votos_nuevos.push_back(votos[i]);
72
73         i++;
74     }
75
76     // requerimos que la imagen este, entonces en este estado i<imagenes.size() e imagenes[i]==imagen
77     votos_imagen = votos[i];
78     i++;
79
80     while (votos[i] == votos_imagen && i < imagenes.size()) {
81         imagenes_nuevas.push_back(imagenes[i]);
82         votos_nuevos.push_back(votos[i]);
83
84         i++;
85     }
86
87     imagenes_nuevas.push_back(imagen);
88     votos_nuevos.push_back(votos_imagen + 1);
89
90     while (i < imagenes.size()) {
91         imagenes_nuevas.push_back(imagenes[i]);
92         votos_nuevos.push_back(votos[i]);
93
94         i++;
95     }
96 }
```

```
100     imagenes = imagenes_nuevas;
101     votos = votos_nuevos;
102 }
103
104 void GaleriaImagenes::eliminarMasVotada() {
105     imagenes.pop_back();
106     votos.pop_back();
107 }
108
109
110 vector<Imagen> GaleriaImagenes::top10 () const {
111     int i = 0;
112     vector<Imagen> result;
113     while (i < 10 && imagenes.size() - i - 1 < imagenes.size()) {
114         result.push_back(imagenes[imagenes.size() - i - 1]);
115         i++;
116     }
117     return result;
118 }
119
120
121 vector<Imagen> dividir (const Imagen &img, int n, int m) {
122     vector<Imagen> resultado;
123
124     int cols = img.anch() / n;
125     int filas = img.alto() / m;
126
127     int i = 0;
128
129     while(i < img.alto()) {
130         int j = 0;
131         while(j < img.anch()) {
132
133             Imagen esta_imagen(filas, cols);
134
135             int i_img = 0;
136             while (i_img < filas) {
137                 int j_img = 0;
138                 while(j_img < cols) {
139                     Pixel este_pixel = img.obtenerPixel(i + i_img, j + j_img);
140                     esta_imagen.modificarPixel(i_img, j_img, este_pixel);
141
142                     j_img++;
143                 }
144                 i_img++;
145             }
146             resultado.push_back(esta_imagen);
147             j += cols;
148         }
149         i += filas;
150     }
151
152     return resultado;
153 }
154
155
156
157 void GaleriaImagenes::dividirYAgregar(const Imagen &imagen, int n, int m) {
158     int alto = imagen.alto();
159     int ancho = imagen.anch();
160
161     if (alto % m == 0 && ancho % n == 0) {
162         vector<Imagen> dividida = dividir(imagen, n, m);
163
164         int i = 0;
165         while(i < dividida.size()) {
166             this->agregarImagen(dividida[i]);
167             i++;
168         }
169     }
170 }
171
172 void GaleriaImagenes::guardar(std::ostream& os) const {
```

```

173     os << "[";
174
175     int i = 0;
176     while(i < imagenes.size()) {
177         if (i != 0) os << ",";
178         os << "(";
179         imagenes[i].guardar(os);
180         os << "," << votos[i] << ")";
181
182         i++;
183     }
184     os << "]" << endl;
185 }
186
187 void GaleriaImagenes::cargar(std::istream& is) {
188     Imagen im(1, 1);
189     int numero_de_votos;
190
191     vector<Imagen> imagenes_nuevas;
192     vector<int> votos_nuevos;
193
194     char charMolesto; // parentesis, coma o chorchete
195     is >> charMolesto; // '['
196
197     is >> charMolesto; // '(' o ','
198     while(charMolesto != ',') {
199         im.cargar(is);
200
201         is >> charMolesto; // ','
202         is >> numero_de_votos;
203         is >> charMolesto; // ')'
204
205         imagenes_nuevas.push_back(im);
206         votos_nuevos.push_back(numero_de_votos);
207
208         is >> charMolesto; // ',' o ']' y se termina el ciclo
209         if(charMolesto == ',') is >> charMolesto; // '('
210     }
211     imagenes = imagenes_nuevas;
212     votos = votos_nuevos;
213 }

```

Código fuente 4: main.cpp

```

1  #include "pixel.h"
2  #include "imagen.h"
3  #include "galeria_imagenes.h"
4
5  #include <string>
6  #include <fstream>
7
8  int main() {
9      string input = "";
10     GaleriaImagenes galeriaCargada;
11     string archivo_galeria;
12
13     while(input != "x") {
14
15
16         cout << "-----" << endl;
17         cout << "| " << endl;
18         cout << "| 1. blur " << endl;
19         cout << "| 2. acuarela " << endl;
20         cout << "| 3. cargar galeria " << endl;
21         cout << "| 4. dividir y agregar " << endl;
22         cout << "| 5. posiciones mas oscuras " << endl;
23         cout << "| 6. top 10 " << endl;
24         cout << "| 7. la mas chiquita con punto blanco " << endl;
25         cout << "| 8. agregar imagen " << endl;
26         cout << "| 9. votar " << endl;
27         cout << "| 10. eliminar mas votada " << endl;
28         cout << "| 11. guardar galeria " << endl;
29         cout << "|-----" << endl;

```

```
30     cout << "Que desea hacer? ";
31     cin >> input;
32     cout << endl;
33     if (input == "1") {
34         int k;
35         string archivo_in;
36         string archivo_out;
37
38         cout << "Ingrese un k, el nombre del archivo de entrada y el de salida" << endl;
39         cin >> k >> archivo_in >> archivo_out;
40
41         ifstream ifs(archivo_in.c_str());
42         ofstream ofs(archivo_out.c_str());
43
44         Imagen im(1, 1);
45         im.cargar(ifs);
46
47         im.blur(k);
48
49         im.guardar(ofs);
50     }
51     else if(input == "2") {
52         int k;
53         string archivo_in;
54         string archivo_out;
55
56         cout << "Ingrese un k, el nombre del archivo de entrada y el de salida" << endl;
57         cin >> k >> archivo_in >> archivo_out;
58
59         ifstream ifs(archivo_in.c_str());
60         ofstream ofs(archivo_out.c_str());
61
62         Imagen im(1, 1);
63         im.cargar(ifs);
64
65         im.acuarela(k);
66
67         im.guardar(ofs);
68     }
69     else if(input == "3") {
70         string archivo_in;
71
72         cout << "Ingrese el nombre del archivo" << endl;
73         cin >> archivo_in;
74
75         ifstream ifs(archivo_in.c_str());
76         galeriaCargada.cargar(ifs);
77         archivo_galeria = archivo_in;
78     }
79     else if(input == "4") {
80         string archivo_in, archivo_in2;
81         int n, m;
82
83         cout << "Ingrese el nombre del archivo de la galeria, el nombre del archivo de la imagen, el n"
84
85         cin >> archivo_in >> archivo_in2 >> n >> m;
86
87         ifstream ifs_galeria(archivo_in.c_str());
88         ifstream ifs_imagen(archivo_in2.c_str());
89
90         galeriaCargada.cargar(ifs_galeria);
91
92         Imagen im(1, 1);
93         im.cargar(ifs_imagen);
94
95         galeriaCargada.dividirYAgregar(im, n, m);
96     }
97     else if(input == "5") {
98         string archivo_in;
99
100         cout << "Ingrese el nombre del archivo de la imagen" << endl;
101         cin >> archivo_in;
102
```

```
103         ifstream ifs(archivo_in.c_str());
104
105         Imagen im(1, 1);
106         im.cargar(ifs);
107
108         vector<pair<int, int> > resultado = im.posicionesMasOscuras();
109         int i = 0;
110         while(i < resultado.size()) {
111             cout << "(" << resultado[i].first << "," << resultado[i].second << ")";
112             i++;
113         }
114         cout << endl;
115
116         galeriaCargada.agregarImagen(im);
117     }
118     else if(input == "6") {
119         string archivo_out;
120
121         cout << "Ingrese el nombre del archivo para el output" << endl;
122         cin >> archivo_out;
123
124         ofstream ofs(archivo_out.c_str());
125
126         vector<Imagen> top = galeriaCargada.top10();
127
128         int i = 0;
129         ofs << "[";
130         while(i < top.size()) {
131             if (i != 0) ofs << ",";
132             top[i].guardar(ofs);
133
134             i++;
135         }
136         ofs << "]" << endl;
137     }
138     else if(input == "7") {
139         string archivo_out;
140
141         cout << "Ingrese el nombre del archivo para el output" << endl;
142         cin >> archivo_out;
143
144         ofstream ofs(archivo_out.c_str());
145
146         Imagen resultado = galeriaCargada.laMasChiquitaConPuntoBlanco();
147         resultado.guardar(ofs);
148     }
149     else if(input == "8") {
150         string archivo_in;
151
152         cout << "Ingrese el nombre del archivo de la imagen" << endl;
153         cin >> archivo_in;
154
155         ifstream ifs(archivo_in.c_str());
156
157         Imagen im(1, 1);
158         im.cargar(ifs);
159
160         galeriaCargada.agregarImagen(im);
161     }
162     else if(input == "9") {
163         string archivo_in;
164
165         cout << "Ingrese el nombre del archivo de la imagen" << endl;
166         cin >> archivo_in;
167
168         ifstream ifs(archivo_in.c_str());
169
170         Imagen im(1, 1);
171         im.cargar(ifs);
172
173         galeriaCargada.votar(im);
174     }
175     else if(input == "10") {
```

```
176         galeriaCargada.eliminarMasVotada();
177     }
178     else if(input == "11") {
179         ofstream ofs(archivo_galeria.c_str());
180         galeriaCargada.guardar(ofs);
181     }
182     else {
183         if(input != "x") cout << "Opcion invalida, por favor seleccione una opcion valida (1-10, x).";
184     }
185 }
186 }
187
188 return 0;
189 }
```

4. Demostraciones

4.1. Invariante de Representación y función de abstracción

InvRep (imp: ClaseGaleriaImagen):

$|imp.imagenes| == |imp.votos| \wedge$
 $(\forall v \leftarrow imp.votos) v \geq 0 \wedge$
 $(\forall i, j \leftarrow [0..|imp.imagenes|, i \neq j) imp.imagenes[i] \neq imp.imagenes[j] \wedge$
 $(\forall i \leftarrow [0..|imp.votos| - 1]) imp.votos[i + 1] \geq imp.votos[i]$

abs (imp: ClaseGaleriaImagen, esp: Galeria):

$mismos(imagenes(esp), imp.imagenes) \wedge$
 $(\forall i \leftarrow [0..|imp.imagenes|]) votos(esp, imp.imagenes[i]) == imp.votos[i]$

4.2. Correctitud del Código

void GaleriaImagenes :: eliminarMasVotada () {

//estado 1;
 //vale InvRep(this);
 //implica abs(this, pre(g));
 //vale $|this.imagenes| > 0$;
 //vale $this == pre(this)$;

imagenes . pop_back ();

//estado 2;
 //vale $this@1.imagenes == this.imagenes + +[this@1.imagenes[|this@1.imagenes| - 1]]$; pues se cumplen los requerimientos de pop_back, dado que $|this@1.imagenes| > 0$
 //vale $this.votos == this@1.votos$;

votos . pop_back ();

//estado 3;
 //vale $this@2.votos == this.votos + +[this@2.votos[|this@2.votos| - 1]]$; pues se cumplen los requerimientos de pop_back, dado que $|this@1.votos| == |this@1.imagenes|$ por invRep(this@1) y a su vez $|this@1.imagenes| > 0$
 //vale $this.imagenes == this@2.imagenes$;

}

//Queremos ver que vale InvRep(this);
 //(1);
 //vale $this.imagenes == this@2.imagenes \wedge this@2.votos == this.votos + +[this@2.votos[|this@2.votos| - 1]]$;
 //implica $pre(this).imagenes == this.imagenes + +[pre(this).imagenes[|pre(this).imagenes| - 1]] \wedge$
 $pre(this).votos == this.votos + +[pre(this).votos[|pre(this).votos| - 1]]$; por las dos proposiciones anteriores y transformacion de estados

//implica $|pre(this).imagenes| == |this.imagenes| + 1 \wedge |pre(this).votos| == |this.votos| + 1$; por propiedad de longitud
 //implica $|imagenes| == |votos|$ pues $|pre(this).imagenes| == |pre(this).votos|$ por InvRep(pre(this))

//(2);
 //vale $this@2.votos == this.votos + +[this@2.votos[|this@2.votos| - 1]]$;
 //implica $pre(this).votos == this.votos + +[pre(this).votos[|pre(this).votos| - 1]]$; por transformacion de estados
 //implica $(\forall v \leftarrow this.votos) v \geq 0$; pues es una sublista de pre(this).votos y vale para esta lista por InvRep(pre(this))

//(3);
 //vale $this@1.imagenes == this.imagenes + +[this@1.imagenes[|this@1.imagenes| - 1]]$;
 //implica $pre(this).imagenes == this.imagenes + +[pre(this).imagenes[|pre(this).imagenes| - 1]]$; por transformacion de

estados

```
//implica  $(\forall i, j \leftarrow [0..|this.imagenes|]) i \neq j \rightarrow this.imagenes[i] \neq this.imagenes[j]$ ; pues es una sublista de  $pre(this).imagenes$  y
vale para esta lista por  $InvRep(pre(this))$ 

// (4);
//vale  $this@2.votos == this.votos + +[this@2.votos[|this@2.votos| - 1]]$ ;
//implica  $pre(this).votos == this.votos + +[pre(this).votos[|pre(this).votos| - 1]]$ ; por transformacion de estados
// implica  $(\forall i \leftarrow [0..|this.votos| - 1]) this.votos[i + 1] \geq this.votos[i]$ ; pues  $this.votos$  es sublista de  $pre(this).votos$ , que esta ordenada
por  $InvRep(pre(this))$ 

//vale  $InvRep(this)$ ;
//implica  $abs(this, g)$ ;

//Ahora queremos ver que valen los aseguras del problema2
// (1);
//vale  $this.imagenes == this@2.imagenes$ ; por transformacion de estados
//implica  $pre(this).imagenes == this.imagenes + +[pre(this).imagenes[|pre(this).imagenes| - 1]]$ ; por pop_back, justificado
en la transformacion de estados
//implica  $[|h| \leftarrow pre(this).imagenes, h \notin this.imagenes] == 1$ ; pues ese elemento que no está es el ultimo, como se ve claramente
en la proposicion anterior, y porque las listas no tienen repetidos, dado que vale  $invRep(this)$  e  $invRep(pre(this))$ 
//vale  $mismos(pre(this).imagenes, imagenes(pre(g)))$ ; por  $abs(pre(this), pre(g))$ 
//vale  $mismos(this.imagenes, imagenes(g))$ ; por  $abs(this, g)$ 
// implica  $[|h| \leftarrow imagenes(pre(g)), h \notin imagenes(g)] == 1$ ; pues usamos los dos ultimos vales ya que iterar sobre listas que son
mismos es lo mismo

// (2);
//vale  $mismos(this.imagenes, imagenes(g))$ ; por  $abs(this, g)$ 
//vale  $mismos(pre(this).imagenes, imagenes(g))$ ; por  $abs(pre(this), pre(g))$ 
//implica  $|imagenes(g)| == |this.imagenes| \wedge |imagenes(pre(g))| == |pre(this).imagenes|$ ;
//vale  $|pre(this).imagenes| == |this.imagenes| + 1$ ; por transformacion de estados y propiedades de longitud, justificado antes
//implica  $|imagenes(g)| + 1 == |imagenes(pre(g))|$ ; por las dos proposiciones anteriores

// (3);
//vale  $pre(this).votos == this.votos + +pre(this).votos[|pre(this).votos| - 1]$ ; por poscondicion de pop_back dado que se cumplen
los requerimientos por los requerimientos de eliminarMasVotada ya que  $|pre(this).votos| == |pre(this).imagenes| > 0$ 
//vale  $pre(this).imagenes == this.imagenes + +pre(this).imagenes[|pre(this).imagenes| - 1]$ ; por poscondicion de pop_back,
por la justificacion de antes
//vale  $sinrepetidos(this.imagenes) \wedge sinrepetidos(pre(this).imagenes)$ ; por  $invRep(pre(this))$ 
//vale  $(\forall i \leftarrow [0..|pre(this).votos| - 1]) pre(this).votos[i + 1] \geq pre(this).votos[i]$ ; por  $invRep(pre(this))$ 
//vale  $(\forall i \leftarrow [0..|this.votos| - 1]) this.votos[i + 1] \geq this.votos[i]$ ; por  $invRep(this)$ 
//implica  $(\forall i \leftarrow [0..|this.votos| - 1]) this.votos[|pre(this).votos| - 1] \geq this.votos[i]$ ; ya que  $this.votos$  esta contenido en  $pre(this).votos$ 
y por  $invRep(pre(this))$  e  $invRep(this)$  ambas estan ordenadas en forma creciente
//implica  $pre(this).imagenes[|pre(this).imagenes| - 1] \notin this.imagenes \wedge$ 
 $(\forall h \leftarrow pre(this).imagenes, h \neq pre(this).imagenes[|pre(this).imagenes| - 1]) h \in this.imagenes$  primera mitad de la con-
juncion por sinrepetidos y el segundo vale y la segunda mitad por el segundo vale
//vale  $(\forall i \leftarrow [0..|pre(this).imagenes|]) votos(pre(g), pre(this).imagenes[i]) == pre(this).votos[i] \wedge$ 
 $(\forall i \leftarrow [0..|this.imagenes|]) votos(g, this.imagenes[i]) == this.votos[i]$ ; por  $abs(pre(this), pre(g))$  ya que se cumple al principio
y  $abs(this, g)$  ya que se demostro previamente que valia el invariante de representacion al final
//vale  $mismos(pre(this).imagenes, imagenes(pre(g))) \wedge mismos(this.imagenes, imagenes(g))$ ; por  $abs(pre(this), pre(g))$  ya
que se cumple el invariante al inicio de la ejecucion
//implica  $mismos(imagenes(pre(g)), imagenes(g) + +imagenes(pre(g))[imagenes(pre(g)) - 1]) \wedge$ 
 $mismos(votos(pre(g)), votos(g) + +votos(pre(g))[votos(pre(g)) - 1])$ ; por  $abs(pre(this), pre(g))$ ,  $abs(this, g)$  y los primeros dos
vales
//implica  $(\forall h \leftarrow imagenes(pre(g))) h \notin imagenes(g) \Rightarrow (\forall j \leftarrow imagenes(pre(g))) votos(pre(g), h) \geq votos(pre(g), j)$ ; si
no pertenece a  $imagenes(g)$  entonces es el que tiene mas votos, se deduce de los primeros dos implica y de  $abs(pre(this), pre(g))$  y  $abs(this, g)$ , ya
que las imagenes de  $g$  y  $this$  (y  $pre(g)$  y  $pre(this)$ ) son mismos y comparten los votos

// (4);
// vale  $pre(this).votos == this.votos + +pre(this).votos[|votos.pre(this)| - 1]$ ; por poscondicion de pop_back, dado que se cumplen
los requerimientos por los requerimientos de eliminarMasVotada ya que  $|pre(this).votos| == |pre(this).imagenes| > 0$ 
//vale  $pre(this).imagenes == this.imagenes + +pre(this).imagenes[|imagenes.pre(this)| - 1]$ ; poscondicion de pop_back, por-
```

²De aquí en adelante, $sinrepetidos(xs : [T]) : Bool = (\forall i, j \leftarrow [0..|xs|], i \neq j) xs[i] \neq xs[j]$

que se cumplen los requerimientos por la misma razón de arriba

// vale $\text{sinrepetidos}(\text{this.imagenes})$; por $\text{invRep}(\text{this})$

//vale $\text{sinrepetidos}(\text{this.imagenes}) \wedge \text{sinrepetidos}(\text{pre}(\text{this}).\text{imagenes})$; por $\text{invRep}(\text{pre}(\text{this}))$ y predicado anterior

// implica $\text{pre}(\text{this}).\text{imagenes}[|\text{pre}(\text{this})| - 1] \notin \text{this.imagenes} \wedge (\forall h \leftarrow \text{this.imagenes}) h \in \text{pre}(\text{this}).\text{imagenes}$; por segundo

y tercer vale

// implica $(\forall j \leftarrow [0..|\text{this.imagenes}|]) \text{this.imagenes}[j] == \text{pre}(\text{this}).\text{imagenes}[j] \wedge$

$(\forall j \leftarrow [0..|\text{this.votos}|]) \text{this.votos}[j] == \text{pre}(\text{this}).\text{votos}[j]$ primer y segundo vale

// implica $(\forall h \leftarrow \text{this.imagenes}) \text{pre}(\text{this}).\text{votos}(h) == \text{this.votos}(h)$; por los 2 implica, los vectores de imagenes y votos de this y

$\text{pre}(\text{this})$ estan ordenados igual y sin repetidos, entonces para todas las imagenes de this.imagenes compartiran los votos

//vale $\text{mismos}(\text{pre}(\text{this}).\text{imagenes}, \text{imagenes}(\text{pre}(g))) \wedge \text{mismos}(\text{this.imagenes}, \text{imagenes}(g))$; por $\text{abs}(\text{pre}(\text{this}), \text{pre}(g))$ y

$\text{abs}(\text{this}, g)$

//implica $\text{mismos}(\text{imagenes}(\text{pre}(g)), \text{imagenes}(g) + +\text{imagenes}(\text{pre}(g))[|\text{imagenes}(\text{pre}(g))| - 1])$;

//vale $(\forall i \leftarrow [0..|\text{pre}(\text{this}).\text{imagenes}|]) \text{votos}(\text{pre}(g), \text{pre}(\text{this}).\text{imagenes}[i]) == \text{pre}(\text{this}).\text{votos}[i] \wedge$

$(\forall i \leftarrow [0..|\text{this.imagenes}|]) \text{votos}(g, \text{this.imagenes}[i]) == \text{this.votos}[i]$; por $\text{abs}(\text{pre}(\text{this}), \text{pre}(g))$ y $\text{abs}(\text{this}, g)$

// implica $(\forall h \leftarrow \text{imagenes}(g)) \text{votos}(\text{pre}(g), h) == \text{votos}(g, h)$; partiendo del implica numero 3 y usando el implica 4 y el anterior

vale llegamos a esto, ya que, por el implica 4 todas las imagenes de g van a estar contenidas en $\text{pre}(g)$ y por el vale anterior sus votos seran iguales para cada imagen de g